Modelling the instrumental value of software requirements

This item was submitted to Loughborough University’s Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: [https://dspace.lboro.ac.uk/2134/21803](https://dspace.lboro.ac.uk/2134/21803)

Publisher: © Richard Ellis-Braithwaite

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: [https://creativecommons.org/licenses/by-nc-nd/4.0/](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Please cite the published version.
Modelling the Instrumental Value of Software Requirements

By

Richard Ellis-Braithwaite

A Doctoral Thesis
Submitted in partial fulfilment of the requirements for the award of
Doctor of Philosophy of Loughborough University
June 2015
© Richard Ellis-Braithwaite
"If you don't know where you are going, any road will get you there."

(Paraphrased from Lewis Carroll)
Abstract

Numerous studies have concluded that roughly half of all implemented software requirements are never or rarely used in practice, and that failure to realise expected benefits is a major cause of software project failure. This thesis presents an exploration of these concepts, claims, and causes. It evaluates the literature’s proposed solutions to them, and then presents a unified framework that covers additional concerns not previously considered.

The value of a requirement is assessed often during the requirements engineering (RE) process, e.g., in requirement prioritisation, release planning, and trade-off analysis. In order to support these activities, — and hence to support the decisions that lead to the aforementioned waste, this thesis proposes a framework built on the modelling languages of Goal Oriented Requirements Engineering (GORE), and on the principles of Value Based Software Engineering (VBSE).

The framework guides the elicitation of a requirement’s value using philosophy and business theory, and aims to quantitatively model chains of instrumental value that are expected to be generated for a system’s stakeholders by a proposed software capability. The framework enriches the description of the individual links comprising these chains with descriptions of probabilistic degrees of causation, non-linear ‘dose-response’ and utility functions, and credibility and confidence. A software tool to support the framework’s implementation is presented, employing novel features such as automated visualisation, and information retrieval and machine learning (recommendation system) techniques. These software capabilities provide more than just usability improvements to the framework. For example, they enable visual comprehension of the implications of ‘what-if?’ questions, and enable re-use of previous models in order to suggest modifications to a project’s requirements set, and reduce uncertainty in its value propositions.

Two case studies in real-world industry contexts are presented, which explore the problem and the viability of the proposed framework for alleviating it. The thesis’ research questions are answered by various methods, including practitioner surveys, interviews, expert opinion, real-world examples and proofs of concept, as well as less-common methods such as natural language processing analysis of real requirements specifications (e.g., using TF-IDF to measure the proportion of software requirement traceability links that do not describe the requirement’s value or problem-to-be-solved).

The thesis found that in general, there is a disconnect between the state of ‘best practice’ as proposed by the literature, and current industry practice in requirements engineering. The surveyed practitioners supported the notion that the aforementioned value realisation problems do exist in current practice, that they would be treatable by better requirements engineering practice, and that this thesis’ proposed framework would be useful and usable in projects whose complexity warrants the overhead of requirements modelling (e.g., for projects with many stakeholders, competing desires, or having high costs of deploying ‘incorrect’ increments of software functionality).

Keywords

Software Requirements, Instrumental Value, Benefits, Wasted Implementations, Goal Modelling, Value Simulation, Requirements Recommendation
Acknowledgements

Without the support, cooperation, and opportunities generously provided to me, this thesis would not have been possible.

I would first like to thank my supervisors, Dr. Russell Lock and Prof. Ray Dawson, for their guidance, patience, and critical thinking. I will be forever grateful for their time and effort spent supporting my work. Prof. Keith Phalp and Dr. Christian Dawson graciously agreed to examine the thesis, and their feedback helped to clarify a number of points.

Secondly, I would like to thank the industrial collaborators for their participation in interviews, surveys, and prototype feedback meetings, for which there are too many to name in totality. Special thanks are due to my industrial supervisors and those who guided my research: Ralph Boyce, Dave Lowe, and Dr. Badr Haque at Rolls-Royce, and Dr. Tim King, William Worsley, Chris Lambert, Dr. Anne Meads, and Dr. James Nyambayo at LSC Group. Of course, thanks are also due to those organisations, who graciously permitted me to spend time located at their offices to benefit from the culture and the valuable experience of their employees. Without them, this thesis’ research questions and answers would certainly have been less ‘real’ and interesting.

Thirdly, to my family and friends, thank you for providing respite from the thesis and for supporting me in its creation. Finally I am able to positively answer the ‘are you finished yet?’ questions! My parents are owed special thanks for nurturing and guiding the choices that lead me to this point.

Thanks are due to the numerous peer reviewers and participants at conferences and meetings for their feedback and discussions, including those otherwise unattributed by this thesis’ publications, such as the BCS Requirements Engineering Specialist Group and Loughborough’s IIS research group. Thanks to the researchers whose work is built upon, and to Scott Adams for the comic relief added to the thesis by his Dilbert cartoons!

Last but not least, thanks are owed to my wife Sachi — the most valuable attainment during the timespan of the project — for proofreading, encouraging, and supporting me through the process.
# Table of Contents

1. Introduction .......................................................................................................................... 1
   1.1 Motivation .......................................................................................................................... 1
      1.1.1 Motivating Requirements Engineering Research .................................................. 2
      1.1.2 Context for this Thesis’ Motivation ......................................................................... 3
      1.1.3 Evidence for the existence of Wasteful FRs/NFRs ............................................... 6
   1.2 Problem Statement & Research Questions ........................................................................... 6
      1.2.1 Thesis Outline ............................................................................................................. 8
   1.3 Contribution ....................................................................................................................... 9
      1.3.1 Novelty ......................................................................................................................... 9

2. Background (Literature Review of Concepts & Problem Evidence) ........................................ 11
   2.1 Conceptual Framework (Concepts and their Relationships) .............................................. 13
      2.1.1 Means & Ends ............................................................................................................ 14
      2.1.2 Cause & Effect ......................................................................................................... 19
      2.1.3 Benefit, Cost, & Value ............................................................................................. 30
      2.1.4 Problems and Risks .................................................................................................. 38
      2.1.5 Desires, Goals, & Objectives .................................................................................. 39
      2.1.6 Requirements (User, Stakeholder, System, Software, etc.) ....................................... 44
      2.1.7 Strategic Alignment .................................................................................................. 48
      2.1.8 Requirements Traceability Links .............................................................................. 50
      2.1.9 Precision, Significance, Accuracy, & Uncertainty .................................................. 50
   2.2 Evidence of the Problem (extended discussion from Chapter 1’s Motivation) ................ 55
      2.2.1 Where did this evidence come from? ....................................................................... 56
      2.2.2 (Theories on the) Causes of the problem ............................................................... 58
   2.3 Context of the Problem ................................................................................................... 59
      2.3.1 When ......................................................................................................................... 59
      2.3.2 Where ....................................................................................................................... 61
      2.3.3 Who ......................................................................................................................... 61
      2.3.4 (For) What ................................................................................................................. 62
   2.4 Summary .......................................................................................................................... 64

3. Research Methodology .......................................................................................................... 65
   3.1 Research Philosophy ......................................................................................................... 67
      3.1.1 Comparing the Rigour of the Research Philosophies .............................................. 69
      3.1.2 This Thesis’ Research Philosophy ............................................................................ 70
   3.2 Research Strategies ........................................................................................................... 70
      3.2.1 Research Strategies for Advancing Requirements Engineering .......................... 71
      3.2.2 Qualitative versus Quantitative, and Inductive versus Deductive ....................... 72
   3.3 Research Design, Methods, and Process .......................................................................... 73
      3.3.1 Research Design ....................................................................................................... 73
      3.3.2 Research Methods (for Data Collection) ................................................................. 76
      3.3.3 Industrial Partners ................................................................................................. 77
      3.3.4 Research Process .................................................................................................... 78
   3.4 Summary .......................................................................................................................... 78

4. Review of Current Value-Based (Means to an End) RE Approaches ..................................... 79
   4.1 Inclusion Criteria .............................................................................................................. 81
1. Introduction

There is often a “field-of-dreams” assumption that once software is built to the specified requirements, benefit will come [Boeh00], or more generally, that “all investments in IT will deliver good results” [Ecka12, p.75]. However, the dangers of this assumption are evidenced by findings of little correlation between a company’s level of Information Technology (IT) investment and its profitability or market value [Carr04, pp.123–124], — a so-called ‘information paradox’ [Thor99].

The act of eliciting, communicating, and analysing what stakeholders want to do with software, i.e., the Requirements Engineering (RE) phase in the software development lifecycle, is intended to reduce the risk that the software will be less useful than it should be.

However, the vast majority of RE activities tend to explore how the software will operate [Berr09], rather than exploring and optimising the software’s value-creating mechanisms. There has often been little incentive to do so, since software engineering has typically occurred in a value-neutral setting [Boeh05]. However, global competition and budget reductions [DeLu14] have amplified interest in more value-oriented approaches to creating and configuring software [AuWo07], as opposed to the prevailing “weeks of coding can save hours of planning” [BDFG05] mentality. Running in parallel is the widespread concern about the extent to which IT is ‘strategically aligned’ [DeLu14] to organisational desires. All such concerns can be summarised by the simple, yet infrequently considered principle that software capabilities are almost always instrumentally good (for their enablement or contribution to other outcomes), rather than intrinsically good (for their own sake).

There is no more appropriate phase in the Software Engineering or procurement lifecycle than RE to explore such concerns. Any earlier risks irrelevance in the analysis, due to uncertainty about the scope of the software-to-be, and any later would incur higher costs in decision changes, as explained by Boehm’s cost-to-fix curve [BoBa01]. It would therefore be logical to assume that, by definition, analysis of stakeholder value is common in RE activities, since:

- benefit is achieved through outcomes that reduce problems [PeWD07];
- goals are prescribed by stakeholders to ‘solve’ problems [Wier09a], and;
- requirements are refined from goals [Lams01].

However, the majority of Requirements and Software Engineering in practice and theory is value-ignorant, and the little work that isn’t is overwhelmingly focused on cost, not benefit [Ecka12, p.22]. Therefore, this thesis proposes a novel framework for applying novel Goal Oriented Requirements Engineering (GORE) techniques in the context of Value Based Requirements Engineering to elicit, model, communicate, challenge, and reason about a software requirement’s benefit creating mechanisms. In order to enable evaluation of the framework, the thesis presents a bespoke software tool to improve the framework’s usability (e.g., using Natural Language Processing to facilitate the re-use of previous project data) and its utility (e.g., using automated graph visualisation and simulation techniques to improve the models’ comprehensibility). Before doing so, existing work on the topic is comprehensively compared and evaluated, and the aforementioned problem(s) are explored with primary research by means of two industrial case studies.

1.1 Motivation

This section provides motivation for research in RE in general (1.1.1), and then for Value Based RE in particular (1.1.1.1). Finally, subsection 1.1.2 clarifies the context and scope of this thesis’ problem.
1.1.1 Motivating Requirements Engineering Research

Brooks famously concluded that “The hardest single part of building a software system is deciding precisely what to build … the product requirements” [Broo87]. Numerous surveys blame the majority of software project failures, including poor return on investment (ROI), on inadequate RE [Euro96, Gart11, HoLe01, Lef97, Stan10, Ukhe03, Vint01] — or more specifically, on poor stakeholder communication [AbPa13] and involvement [JoHo06], and incorrect assumptions [GiCo08]. As Finkelstein succinctly put it: “Whenever practitioners are questioned about difficulties in system development they stress inadequate requirements engineering as a major cause of problems” [Fink94]. As a result, many RE researchers have thought it “unnecessary to set down an extensive motivation for research in requirements engineering” [Fink94]. Indeed, since the quality of a system is determined by the degree to which it meets stakeholder requirements, improving the requirements and the efficiency with which they are engineered is the ‘most direct’ route to “reducing the overall cost of software, improve its quality, and dramatically shorten the time to market” [MaTh13a]. Finally, van Lamsweerde motivates RE research with a controversial yet pertinent question: “Given the expected progress in component reuse and automated programming technologies, will there be anything else left in software engineering, beside software geriatry, than requirements engineering?” [Lams01].

The Standish Group’s bi-annual CHAOS survey on software project success is the largest and most cited, having captured data from over 90,000 IT projects. Recent results revealed that [Thes13]:

- 39% of software projects were considered to be successful in scope, budget, and schedule;
- 43% completed but ‘with partial functionalities, major cost overruns, and significant delays’ (within this category, 74% of the projects overran on time, 69% did not meet the specification, and 59% overran on cost [Thes13, p.2]);
- 18% failed to complete.

The majority of these software project failures and challenges tend to be attributed to RE inadequacies [Gart14], (specifically, ‘lack of user involvement’, ‘requirements incompleteness’, ‘changing requirements’, ‘unrealistic expectations’, and ‘unclear objectives’ [Lams09a, p.48].) The blame on RE has not changed over time, as shown by the European Software Institute’s two decade old survey [Euro96] (amongst others). However, RE’s culpability for project challenges is not by itself sufficient motivation for research in the field. Indeed, a number of renowned RE researchers have recently called for radical changes in future RE research [MaMa13, Maid11, Wier05]. Consequently, generic ‘RE is important, our work is on RE, ergo our work is important’ motivations are dissuaded due to the low uptake of RE research in practice [MaLa08, NeLa03, SABP07]; Given that adherence to good requirements engineering practice is neither necessary nor sufficient for software project success [DaZo06], it is important to contextualise the type of problems that RE research is focused on to ensure its validity in the real world, as the next Section (1.1.2) sets out to do. First however, is a brief motivation for research in Value-Based Requirements Engineering (this thesis’ field).

1.1.1.1 Motivating Value-Based Requirements Engineering Research

Studies of IT project failure have consistently shown the importance of managing benefits, as opposed to the traditional schedule, budget, quality & scope ‘iron triangle’ indicators [PyOl13]. Lin and Pervan’s review went as far as claiming that the number one cause of project failure is “the vague statement of benefits, leading to an uncertain allocation of responsibility for managing their delivery” [LiPe01]. Similarly, studies of the top risks in software engineering [Boeh07, Jone94, Rei02] show that process guidance related deficiencies far outrank technological deficiencies. (To illustrate, the most common risks cited by these sources are ‘low user satisfaction’, ‘creeping user requirements’,
More specifically within RE, numerous studies have shown that most significant RE errors tend to be caused by incorrect assumptions about the problem world, rather than about the machine (i.e., software) to be built [Lams09a, p.49]. In addition to academic reports on the importance of value considerations in requirements engineering [AuWo07], practitioner oriented reports have both historically [KaMZ06, Whit99] and recently [DoHo06, Pano10] stressed the causal correlation between effort spent on benefit analysis, and software project success.

Yet, academic research in software requirements engineering has tended to focus more on technical aspects, e.g., “on new languages and sets of notations” rather than for “guid[ing] requirements engineers in the incremental elaboration and assessment of requirements” [Lams01]. More than 10 years have passed since that observation, and there is still a mismatch between RE research and the needs of RE practitioners [JLLM11, Maid11, MaMa13]. In particular, very little has focused on guiding researchers to understand stakeholder value concerns, which as Value Based RE researchers argue, is a considerably more prevalent concern in the majority of modern software projects [Jain07] than those typically held by software engineering researchers (e.g., on proving program correctness).

1.1.2 Context for this Thesis’ Motivation

This thesis is motivated by software projects whose resulting capabilities failed to be as beneficial as expected, and where ‘better’ RE could have reduced or avoided this (subset D in Figure 1.1).

![Euler diagram showing the type of RE failure motivating this thesis (subset D)](image)

**Software Project Failures (A)** occur where a significant aspect of the project was less than desirable. Popular definitions of ‘success’ within industry surveys [EvVe10] use the traditional ‘iron triangle’ variables. I.e., the project would have failed in some way if it took too long (time), cost too much (money), or if it produced insufficiently capable software (product scope/quality) [Ieee11a].

**RE Failures (B)** occur where the stakeholder’s actual problem is different to the problem described at the RE stage, or where the abstract solution prescribed at the RE stage is sub-optimal for solving the problem (building on Wieringa’s two ‘schools of thought’ RE definition [Wier05]). RE specification failures of particular interest therefore include: ‘obsolescence’ [WnGZ13], ‘gold plating’ [Wieg90, pt.5], or lack of ‘validity’ [Clel05, p.10], ‘pertinence’, or ‘adequacy’ [Lams09a, p.35]. Failures in artefacts preceding requirements are included in this set (B), e.g., if a requirement’s existence is motivated by some expected benefit that is unattainable. Failures of artefacts resulting from requirements are excluded by this set (B), e.g., bad prototypes or software that fails to meet the specification.

**RE Failures contributing to Software Project Failures (C)** represent only a subset of RE failures that this thesis is interested in, since not all RE failures cause a software project failure. For example, recent studies of particular requirements specification defects, such as ambiguity, show no
significant correlation with project success [PHKC13]. (Nevertheless, a significant body of RE research aims solely to reduce specification defects, such as ambiguity, completeness, or consistency — see [Fire05] for a more complete enumeration of specification defects.)

**RE Failures contributing to Software Project Benefit Realisation Failures (D)** represent the primary focus of this thesis: RE failures that have lead to a set of software capabilities that has failed to be as valuable as the stakeholders expected. This is primarily indicated by failures in software product quality or scope, but it may instead manifest as failures in cost or schedule. Such a scenario might be that, (in the context of iterative software development) an unnecessarily high number of development iterations were required in order for the ‘correct’ software capability to be understood. Unfortunately, this might only be perceived as a development, methodology, or scheduling problem, — rather than as an RE problem — especially by those removed from the engineering process, such as organisational managers. (Interestingly, ‘traditional’ cost/schedule failures have received significantly more proactive, reactive, and evaluative attention in practice than benefit realisation failures [Bone11, p.11, SaHK08, p.63].)

In more focused terms, this thesis’ problem is defined by two failures (visually classified in Table 1.1):

- Implementing requirements that lead to wasteful or inadequate software capabilities, and;
- Not implementing requirements that would have been valuable.

In other words: where requirements are overvalued, and where requirements are either undervalued or unknown. Both of these failures could be considered as types of waste (as in ‘Lean Thinking’ [PoPo03, WoJo03]), since either development resources are wasted, or opportunity for benefit is wasted. The remainder of this section describes how these failures may lead to waste.

**Table 1.1: Success or Failure Classification: Software Requirements — Implemented vs. Useful**

<table>
<thead>
<tr>
<th>Requirement Implemented?</th>
<th>Resulting Software Capability Sufficiently Useful?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>✔ Success (#1)</td>
</tr>
<tr>
<td></td>
<td>✗ Failure (#3)</td>
</tr>
<tr>
<td>No</td>
<td>✗ Failure (#2)</td>
</tr>
<tr>
<td></td>
<td>✔ Success (#4)</td>
</tr>
</tbody>
</table>

Implementing functionality later perceived not to be sufficiently useful (#2), could result in it being:

i) **Scrapped**: the functionality may be removed, leading to more development work required to fulfill the actual need (budget, schedule, and development capability permitting);

ii) **Re-worked**: the functionality may be significantly modified to address the actual need;

iii) **Unused**: the functionality may remain in the system, never to be used, potentially leading to ‘bloated’ user interfaces [Mcgr00] and consequences of redundant code [Arse09, p.110], such as:

- waste in creating, testing, maintaining, and documenting the code;
- waste in memory and bandwidth for compiling, storing, and distributing the code;
- waste in future efforts if new product requirements are derived from the code.

(Since they may not be obvious to the reader, some reasons for leaving unused functionality in software are provided: Developers may not know which functionality is useful and which is not, since not all types of software provide an easy mechanism for recording and communicating usage statistics (e.g., like HTTP server log files do); Developers might hope that their work would later be useful, rather than admit fault and discard their work; The cost of storage space for dead code is in most cases trivially small; or, Developers may be unwilling to remove the functionality for fear that they may disrupt a user’s workflow, especially if users have ‘re-imagined’ uses for functionality, i.e. where (iv) occurred.)

iv) **Re-imagined**: the functionality may remain in the system, not fulfilling its intended ‘usefulness’. Eventually, users may find some unintended uses for it, but that would be due to good luck rather than intent, and should not be relied upon to offset its development costs.

4 1.1 Motivation
Conversely, not implementing potentially useful functionality (#3) could affect the software’s:

i) **Scope:** the software may be ‘incomplete’ if it does not deliver sufficient utility, leading to;

ii) **Schedule:** more work (time) may be required to fix the above scope-problems, leading to;

iii) **Cost:** more money may be required to fix the above (as well as costs to reputations/careers).

On balance, these latter consequences of not implementing requirements (failure #3) are likely to be the more important of the two failures, since they present the most direct obstacle to the goal of satisfying the stakeholders. Unfortunately, this failure (#3) is also likely to be the less visible of the two, since it is an error of omission rather than of commission.

1.1.2.1 **Context for this Thesis’ Motivation: Viewpoints**

Finally, there are two possible viewpoints to observe the two failures that motivate this thesis: from the means viewpoint (the software), or from the end viewpoint (the benefits expected from the software). For example, if a software capability is not useful post-implementation, then you might observe from the means viewpoint that the functionality is not used frequently. From the ends viewpoint, you might observe that the problem that originally motivated the functionality is unaffected or unimportant.

Figure 1.2 and Figure 1.3 illustrate the two failures from the means viewpoint: Figure 1.2 for functional requirements, and Figure 1.3 for non-functional requirements. (Note that the curves in both figures are only intended to be representative. In actuality, the curves would vary amongst individual software functionalities and qualities.)

Figure 1.2: Waste indicated by redundant implemented functionality (left: failure #2), or by unimplemented functionality (right: failure #3).

Figure 1.3: Waste indicated by superfluous quality (left: failure #2), or as inadequate quality (right: failure #3).

Figure 1.2 has the three noteworthy caveats. Firstly, frequency of a software feature’s use is a strongly correlated and accessible indicator of its value (the real concern), but it does not determine the
value of the functionality. E.g., a ‘restore backup’ function might be used only once with huge benefit, while a ‘print’ function might be used frequently for no real purpose other than habit. Secondly, infrequently used functionality might not be considered to be a problem in market driven (compared to bespoke) software development, since large feature sets may help to sell the product. Thirdly, infrequent usage may be due to poor usability, rather than poor utility. However, this could be framed as an RE problem, e.g., by specifying usability requirements or other means (such as user training) as necessary for realising a requirement’s potential utility.

Figure 1.3 has the caveat that quality increases tend to increase development or procurement costs, however, if there are no costs in achieving superfluous levels of software quality, then it would not be wasteful. In such a case, Figure 1.3’s curve would start at high utility, rather than low utility.

(Figures showing utility curves from the ‘ends viewpoint’ are not included, since in most cases, utility would monotonically increase as the satisfaction of the end increases, and failure is simply low utility.)

1.1.3 Evidence for the existence of Wasteful FRs/NFRs

Existence of these RE failures is evidenced by studies and surveys in the literature, albeit sparsely and rarely with an RE focus. Borg et al. remark that it is “a widely accepted fact that obtaining feature usage data is very difficult” [BKOS04]; While predictive models of feature usage can be found (known as ‘operational profiles’ [Kozi05] or ‘functional profiles’ [Musa93]), the literature makes far fewer descriptions of actual (historic) usage available. (Such models are very rarely used within RE, instead being popular in the usability field [ReKG04, p.10]). Remarking on the dearth of empirical evidence to quantify failure in software engineering, Glass summarises that “At first glance, there are plenty of publications that conclude there really is such a [software] crisis … Most such academic papers and guru reports cite the same source for their crisis concern—a study published by the Standish Group” [Glas06]. Hence, it might not be surprising to the reader that the Standish Group’s CHAOS report also provides the primary evidence for the existence of implemented-but-not-useful-functionality, claiming that a surprisingly large “50% of features are hardly ever or never used” [Thes13, p.2]. No other large-scale evidence appears to be available, other than a combined set of anecdotes, surveys, and case studies from widely varying contexts — perhaps primarily because it takes a great deal of effort to build awareness and trust with industry to run such surveys. Nevertheless, the CHAOS findings are generally accepted because practitioners can relate their own experiences to them, and because in RE practice, stakeholders are often “motivate[d] to brainstorm requirements which they think that they just might need at some point, guaranteeing that a large percentage of the functionality that they specify really isn’t needed in actual practice” [Ambl12].

A more detailed evaluation of the literature supporting the existence of the problems that this thesis is primarily interested in (i.e., failures #2 and #3 in Table 1.1), is provided later in Chapter 2.2.

1.2 Problem Statement & Research Questions

The overarching problem statement is defined as the following broad research question:

How can the instrumental value of software requirements be modelled before they are implemented, so that the resulting software’s usefulness can be optimised?

1 In essence, “‘X has instrumental value’ means X is conducive to something else that has value” [Bear65, p.5]. (The concepts in this statement are more thoroughly defined, and related to others, in Chapter 2.)
2 A model is a mutable [Roth89] simplification [BeBG05] that allows one to reason about the subject [BoBo07].
3 As per the dictionary definition, useful is considered synonymous with beneficial or valuable [Coll14].
This is refined into five research questions in Figure 1.4, that are again refined into more-operational questions (i.e., from aims to objectives), such that answering them implies completeness of the thesis. (While the general research questions were identified at the beginning of the project, the more specific research questions were refined as the project progressed. Thus, Figure 1.4 represents the final state of the project’s decomposition, and not necessarily the original plan.)

Figure 1.4: Research process for answering this thesis’ research questions. (The distinction between Knowledge Problems (KP) & Design Problems (DP) is discussed in Chapter 3.)

Stakeholders responsible for a software project’s continued funding need to be satisfied with the level of risk that the software-to-be may not be as beneficial as expected or as is possible, and that its promised capabilities will not be redundant. Software is an aggregation of interacting but individual functionalities and their qualities, and so value models should have granularity at the level of requirements rather than projects. The mechanisms by which these requirements would
create benefit are inherently uncertain (being future phenomena), and so should be communicable for risk assessment and reduction. Finally, stakeholders performing RE activities where the benefit of a requirement is questioned (e.g., in prioritisation, release planning, trade-off, etc.) are required to understand how benefit is defined by the stakeholders, and then how the requirements (and their alternatives) would contribute to the realisation of that benefit, and with what degree of confidence.

These ideals are unfulfilled by the state-of-the-art RE approaches, and especially in current practice (as is argued later in Chapter 4’s literature review and Chapter 6’s case studies).

1.2.1 Thesis Outline

Chapter 1: Introduction – Introduces, clarifies, and motivates the research problem. The problem is defined in terms of answerable research questions. A summary of the contribution that the thesis makes is described.

Chapter 2: Background – Presents the conceptual framework for the thesis, providing a synthesis of the core definitions and relations for the problem’s underlying concepts. The context of the problem is explained in order to describe the when, where, who, why and how aspects, and evidence from the literature to support the existence of the problem is discussed and evaluated.

Chapter 3: Research Method – Defines the methodologies by which the research questions will be answered, starting at problem exploration with structured interviews and questionnaires, through to the design and evaluation of the main artefact (the framework) with industry case studies.

Chapter 4: Review of the State of the Art Approaches – Defines the requirements for an approach that would aim to reduce the problem identified in Chapter 1. Classifies, summarises, evaluates, and compares the state of the art approaches to the requirements, in order to identify the gap, i.e., the key area of focus for the framework’s development, and hence this thesis’ novelty.

Chapter 5: The Framework – Describes the framework designed to satisfy the requirements outlined in Chapter 4, which is an integration of agent based goal modelling, system dynamics modelling, benefits management, expert estimation, uncertainty quantification, and automated information retrieval and visualisation techniques. Goal modelling was selected for the foundation of the framework because the RE community has accepted it as the de facto standard for intentional modelling, i.e., for modelling ‘why?’ a system’s capabilities and qualities should be chosen over their alternatives [Lams01] (Chapter 4 discusses this in far greater depth). Finally, Chapter 5 provides a reference implementation, and presents the software tool created to support the framework’s usage.

Chapter 6: Case Studies – Firstly presents the results of primary research in two industry case studies in order to understand the problem’s incidence, causes, and consequences. Secondly it presents an evaluation of the framework using expert opinion, in order to motivate the approach’s utility and usefulness relative to the requirements set out in Chapter 4. Finally, it summarises the findings on the problem, as well as the effects of implementing the approach in the case studies, providing guidelines for the approach’s implementation, and other lessons learned.

Chapter 7: Conclusion – Concludes with the contributions of the thesis and areas of future work. Table 1.2 maps the research questions to thesis chapters in order to aid traceability and navigation.
Table 1.2: Mapping research questions to chapters. (Key: {✔ Strongly, ✅ Somewhat} answered).

<table>
<thead>
<tr>
<th>Chapter</th>
<th>RQ1</th>
<th>RQ2</th>
<th>RQ3</th>
<th>RQ4</th>
<th>RQ5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1 – Introduction</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 2 – Background (Conceptual Framework)</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 3 – Research Methodology</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 4 - Review of State of the Art Approaches</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 5 – Proposed Framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Chapter 6 – Case Studies (Problem &amp; Solution)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Chapter 7 – Conclusion</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

1.3 Contribution

The contributions of this thesis are twofold, firstly through new knowledge, and secondly through a new design artefact:

1. An exposition of requirements engineering in terms of assuring instrumental value via causal reasoning, resulting in a novel conceptual framework synthesised from wide ranging literature.

1.1) A comparison and evaluation of the seminal and current approaches suitable for modelling the instrumental value of software requirements. This allows wide ranging techniques to be understood from a common perspective, highlighting overlaps and gaps in the literature.

1.iii) Industrial case studies on the incidence and consequences of redundancy in software caused by inadequate RE, both supporting and conflicting with existing conclusions on the subject.

2. A tool-supported framework\(^4\) to elicit and quantitatively model and simulate (i.e., answer ‘what if?’ questions about) the alignment of software requirements to their sources of value, in order to improve value-oriented RE decisions, resulting in a novel artefact for RE practice.

Note that the key contributions (the framework, the software tool acting as a proof of concept, and the case studies) aided each other’s development cyclically rather than unidirectionally. For example, the development of the software tool by the researcher has matured the framework by bringing operational issues into light. Chapter 3 discusses these feedback loops in more depth.

1.3.1 Novelty

To the best of this researcher’s knowledge (and the opinions of the peer reviewers of the published papers), prior to this thesis, an approach did not exist in the literature that enabled the effect of a software requirement’s satisfaction on its intended benefits to be modelled and analysed:

a) at varying levels of goal satisfaction extent (explaining the effects of partial/full requirement satisfaction on multiple levels of goal abstraction);

b) at varying levels of stakeholder confidence (explaining the extent to which a requirement’s satisfaction may not contribute to a goal as specified);

c) at varying levels of software usage (explaining the different profiles of soft-ware usage that could affect a requirement’s contribution to higher level goals);

d) at varying levels of stakeholder utility (explaining the non-linear relation-ships between the extent of a goal’s satisfaction and the stakeholder satisfaction to be gained);

e) at varying levels of stakeholder agreement (explaining the variance between the stakeholders’ estimates about the benefits that will be contributed);

\(^4\) The tool is available to download at http://richardeb.com/gviz along with tutorials and example files.
f) with limited prior knowledge about the business goals (assisting the elicitation of the intrinsic values that the software features are intended to be instrumental for realising);
g) with tool support to describe, simulate, visualise, validate, communicate, re-use, recommend, and assist the modelling of a software requirement’s value creation mechanism(s).

Furthermore, reasons for and consequences of wasteful software requirements had not been studied in detail or in particular application contexts (as they are in Chapter 6), prior to this thesis. Additionally, the degree to which current RE practice adequately describes a software project’s problems and solutions had not been studied (as studied in one of this thesis’ papers [ELDK14]).
2. Background (Literature Review of Concepts & Problem Evidence)

“There’s no sense in being precise when you don’t even know what you’re talking about.”

John von Neumann
Firstly, this chapter establishes this thesis’ conceptual framework (Section 2.1), which defines the core concepts and their relationships, so as to enable a shared understanding and hence avoid ambiguity in the reader’s interpretation of the research questions and their resulting work. A conceptual framework is a key output in design science research, and is also known as a model of constructs [HMPR04, RoSe03]. They are advocated in checklists for software engineering research [Wier12], since they help to ensure ‘construct validity’ [ESSD07]. It is, by itself, a novel contribution as a synthesis of the literature on the underlying concepts in modelling a requirement’s value. While some of the content in this chapter may seem theoretical or philosophical, it does have practical implications for the framework, which might otherwise have been ‘learned’ from naïve practical experimentation. Indeed, much of it represents ‘lessons learned’ from implementing the framework.

A conceptual framework is especially important for this thesis, given that it concerns the alignment of ‘software requirements’ and ‘value’—conceptual things, rather than tangible things. If these two concepts are not clearly defined amongst the concepts they depend on, then the thesis could be interpreted by readers in different ways according to their own understandings. This variety of understandings for a given concept requires that the conceptual framework’s scope is defined: It is intended to be the ‘single version of the truth’ within the bounds of this thesis. So, for example, if a different understanding exists in an organisation, then neither of the understandings are necessarily ‘wrong’. The ‘correct’ understanding is the most useful in that particular context, since RE is a practical pursuit intended for maximising utility, rather than wholly a scientific pursuit for the ‘truth’. 

Secondly, the chapter expands and evaluates the literature that provides evidence for this thesis’ problem (Section 2.2). Thirdly and finally, the context of this thesis’ problem and for its framework (i.e., the when, where, who, and what aspects) is described in Section 2.3.

This chapter is not to be confused with the review of the state of the art approaches applicable to the thesis’ problem. This is located later in Chapter 4, since the reader’s interpretation of Chapter 4’s inclusion and comparison criteria should be influenced by this chapter’s conceptual framework.
2.1 Conceptual Framework (Concepts and their Relationships)

Figure 2.1 relates the concepts detailed in this chapter using the Unified Modelling Language (UML). Ontologically speaking, it is not exhaustive, and is under the ‘open-world assumption’ [ZeLu12, p.5].

(UML’s Class Diagram notation is used in Figure 2.1 because it is the standard language used to “visualize, specify, construct and document” domain knowledge in the software engineering community [Chan01], and because it can be straightforwardly translated to explicit ontologies [ZeLu12], e.g., to a...
formal language such as OWL2. Furthermore, unlike ontology languages, UML has a standard graphical representation, a very large user community, and a higher likelihood of practitioner-familiarity than the alternatives [CrPu99].)

2.1.1 Means & Ends

The underlying theme of this thesis is that a software requirement is a means to an end, dependent on a chain of other means and ends for it to be valuable. Therefore, in analysing the benefit of a software requirement, it is essential to understand the often numerous ends that motivate its existence. Without doing so, software could not be ‘valid’, but only ‘verified’ [Boeh84], (where valid software is able to achieve the desired ends, and verified software is a good quality means — e.g., lacking bugs). Since software may contribute to an end-user’s desires, but not those of their organisation, then eliciting, prioritising, and rationalising a complete set of ends to evaluate the software’s ‘goodness’ is crucial to the validation exercise (both after and before implementation), and hence ultimately the perceived success of the software.

The concept of a ‘means to an end’ is at least 2000 years old, originating from Cicero’s ‘principle of selection’: “[T]he wise man... endures pains to avoid worse pains” [Cice45], (i.e., un-pleasurable things should be endured only as a means to an end —more pleasurable things). Plato similarly proposed a distinction between “things that are to be welcomed solely for themselves”, i.e., ‘intrinsically good’ things (ends in themselves), and “others that are to be pursued only for their results”, i.e., ‘instrumentally good’ things (means to ends) [Krau10]. More recently, a ‘means to an end’ has been defined as “something that you are not interested in but that you do because it will help you to achieve something else” [Camb06]. Hence, if software is to be considered ‘a means to an end’, we must accept the assumption that when implementing or acquiring software, we are not interested in it “for its own sake, ...in itself, ...as such, ...in its own right, ...without qualification, ...absolutely, ...tout court, ...or sans phrase” — i.e., because it is intrinsically good [Krau10].

The reader may reject this assumption since many people are seemingly interested in software for its own sake. However, Moore's isolation thought experiment [Moor05, p.28] should be considered to better understand intrinsic value and its rarity: The intrinsic value of a thing does not depend on anything else, so it cannot be destroyed by the removal of everything else. Hence, Moore asks ‘would you find value in the supposedly intrinsically good thing in a universe containing only that thing in absolute isolation?’ When asked about software, the answer would most likely lead to the conclusion that interest in software ‘for itself’ is instead interest in it as a hobby (for pleasure), for task automation (to create free time), for career advancement (to make money), and so on.

Summary Point #1: Implication for the Thesis

Software is valued not for what it is, but for what it does. Hence, the goodness, value, or benefit of software is instrumental.

2.1.1.1 Means & Ends: Intrinsic versus Extrinsic Value

A means to an end is instrumentally valuable by definition, and so derives its value solely from its end. However, this end must ultimately derive its own value from a source of value not dependent on another end [Zimm10]. The reader can prove this for themselves: Imagine being asked ‘is it good to be kind to others?’, and then for every answer you provided, the inquirer asked ‘what is good about that?’ Eventually, you would be forced to realise that if one thing derives its value from another thing, which derives its value from another thing, and so on, then logically there has to be a point where the value generated is not derivative, otherwise nothing would have any value [Zimm10]. (In the words of Parmenides, “nothing comes from nothing” [Roec10, p.37], and therefore
“something must be desirable on its own account” [Hume57, p.111].) Hence, to understand how a software feature derives its value through constructing chains of means and ends, it is useful to know when an end’s source of value is not instrumental, but extrinsic, intrinsic, instrumental, or final.

**Extrinsic value** is a general class of value opposite to intrinsic value, often confused with its more specific subtypes [Olso67]: instrumental value (a particular way of causing other value, e.g., MySQL for storing data) or contributory value (value dependant on being part-of-a-whole, e.g., a line of code in a function). In essence, extrinsic value exists by virtue of something’s relational properties, rather than in its non-relational properties [Kors83]. For example, the pen that Lincoln used to sign the Emancipation Proclamation derives its value in virtue of an extrinsic property: belonging to Lincoln [Kaga98a]. For this reason, extrinsic value is sometimes referred to as ‘parasitic’, in that if the “extrinsic value of A can be traced to the intrinsic value of Z by way of B, C, ..., Y”, as in Figure 2.2, then A-Y are said to generate no value of their own. Therefore, “no change of value would be effected in or imported to the world if a shorter route from A to Z were discovered” [Zimm10]. In other words, a stakeholder may claim to desire Y, but Y might be unnecessary if Z can be achieved in other ways.

Figure 2.2: The shortest path between wholly extrinsic value and its source of intrinsic value is theoretically equal in value to the longest path

**Intrinsic value** (otherwise known as ‘terminal value’) can most clearly be defined as value in itself, in virtue of its non-relational properties [Kors83], where non-relational properties are those inherent and not dependent on any other entity (e.g., being loved is relational, while being male is non-relational). Frankena has proposed perhaps the most comprehensive list of intrinsically valuable things: life, consciousness and activity; health and strength; pleasures and satisfactions of all or certain kinds; happiness, beatitude, contentment; truth; knowledge and true opinions of various kinds, understanding, wisdom; beauty, harmony, proportion in objects contemplated; aesthetic experience; morally good dispositions or virtues; mutual affection, love, friendship, cooperation; just distribution of goods and evils; harmony and proportion in one’s own life; power and experiences of achievement; self-expression; freedom; peace, security; adventure and novelty; and good reputation, honour, and esteem [Fran73]. It is theorised that these values were “forged by evolution in the ancestral environment to maximise inclusive genetic fitness”, and so they too were once extrinsically valuable [Less14]. Unfortunately (for the sake of their validation), humans cannot introspect their values, and to add to their complexity, they are often volatile, contradictory, and above all, personal. Hence, an infallible universally applicable set of values from which requirements could be derived or conflicts resolved, cannot exist. Furthermore, they cannot be compressed or summarised by rules, in the same way that “dialling nine out of ten digits correctly may not connect you to a person who is 90% similar” [Yudk11, p.6]. Consequently, except for acting as a stopping point for instrumental value exploration, such a list of intrinsic values as Frankena’s would not be of practical use in RE without the construction of a consistent utility function [Yudk11], requiring the elicitation of a stakeholder’s own preferences over each value — much like requirements prioritisation, but at the highest possible level of desire abstraction. (Note that researchers have attempted to define a generic order of preferences for a
smaller subset of such values. E.g., Maslow’s hierarchy [Masl54], as discussed in Section 2.1.3.3, provides rationale for assigning higher utility to a software requirement ensuring ‘health’ than one ensuring ‘good reputation’, for example.)

Frankena’s list encapsulates the answers of almost every philosopher’s attempt at answering the question of ‘what has intrinsic value?’ [Fran73], but for that fact alone it is not universally accepted. For example, Hedonists believe that only pleasure has intrinsic value. Meanwhile, others have questioned whether it is possible to be intrinsically good, since we cannot know for certain if we have experienced intrinsic value or not [Bear65, p.12] (although this argument is often rejected since not knowing something does not preclude its existence). Perhaps the most convincing argument against the existence of intrinsic value is offered by Dewey’s theory of pragmatism [Dewe22], in that what is an end in one context is a means in another, and so it is a mistake to seek wholly intrinsic goods or evils. Indeed, since the debate around the existence of intrinsic value started in the era of Plato and Aristotle, and is still ongoing [Krau10], Zimmerman succinctly concludes that “It may well be that, the world being as complex as it is, nothing is such that its value is wholly intrinsic; perhaps whatever has intrinsic value also has extrinsic value, and of course, many things that have extrinsic value will have no intrinsic value” [Zimm10].

To conclude on the concept of intrinsic value for the context of this thesis, it is proposed that a chain of ‘what is good about that?’ questioning starting at a software requirement (‘A’ in Figure 2.2) will lead to something more desirable in isolation (i.e., valuable not for its relational properties) than the software itself (‘B(Y’ in Figure 2.2). In turn, this source of less extrinsic value will ultimately derive its value from a terminal source of value least dependent on other sources of value (‘Z’ in Figure 2.2). In all likelihood, Z will represent one of Frankena’s intrinsically valuable things in one form or another, but whether Z is ‘purely’ intrinsic is an argument beyond the scope, and interest, of this thesis. Hence, for the context of Requirements Engineering — a pragmatic and problem solving discipline, this adaptation of the pragmatic view on intrinsic value seems appropriate. Besides, by the time something that could be considered mostly intrinsically good is reached in such a chain, the requirement’s benefit creating mechanism would likely have been explained transparently enough to understand ‘the essence of the requirement’ and the ‘win condition(s)’ for the requirement’s stakeholder(s) [BBHL94].

**Summary Point #2: Implication for the Thesis**

All sources of instrumental value generated by a software capability will ultimately terminate at a source of intrinsic value, hence providing a natural boundary for stopping to ask “why implement this?” questions (to sufficiently explore the value).

Of perhaps more significance to this thesis is that one person’s intrinsic good may be extrinsically good for another person. Take, for example, the popular business mantra ‘Happy People Sell’ (formally known as ‘positive psychology’ [Acho10]), where an employee’s happiness (intrinsically good for themselves) may be extrinsically valuable for the organisation’s total sales, which is good for the shareholders’ personal finances, for the government through taxation, and so on. (Though, RE models need only focus on a subset of people who would be affected — as elaborated in Chapter 5.) In the RE literature, rather than viewing this as a network of means-ends chains, it is more often considered as a network of actor dependencies [Yu97] or stakeholder responsibilities [Somm07a].

**Summary Point #3: Implication for the Thesis**

IT is often used in such a way that it benefits the owner’s organisation, and so chains of value derivation for sociotechnical software capabilities will likely pass through numerous value recipients (each link providing value for a person or group).

Contrary to the notion of exploring the instrumental value of a software requirement is Zave and Jackson’s criticism of goal abstraction as a starting point in RE, which argues that goal (and hence
requirement) elicitation must be bounded by the system’s subject matter [ZaJa97, p.5]. They present the example of a zoo turnstile system development project, and stipulate that requirements engineers should not propose that the zoo should be sold to increase the owner’s profits — which is stated as the goal of the turnstile system via avoidance of non-paying entrants. Kovitz similarly asserts that software requirements documents should not document an open-ended problem such as to “increase sales” or “help the marketing department improve the web pages” [Kovi98, p.31]. The argument is that programmers will implement requirements by making software, and so “the only type of problem we’ll describe in a software requirements document is a software problem”, e.g., of being able to “provide reports showing the number of hits to the web page” [Kovi98, p.31].

However, such exclusive focus on what the software should be directly responsible for, goes against the ‘dependency theory’ component of ‘Value-Based Software Engineering’ (VBSE) [BoJa06, Jain07], by not defining the software initiative(s) in their context. Indeed, even outside of the VBSE field, developers and architects are considered to be responsible for “finding out the real story behind the requirements” [ClBa10]. Focusing purely on the ‘software problems’ assumes that either the software capabilities will be intrinsically good, or that they will be extrinsically good, but that there is no need to consider what in particular the software capabilities would be good for.

Continuing the zoo example, perhaps it is the case that the turnstile system is not a sufficient strategy for satisfying the profitability goal of the zoo owner due to incorrect assumptions, or ‘all else not being constant’ in the world. E.g., a reduction of non-paying entrants by itself will not increase the revenue of the zoo, but rather a conversion of non-paying entrants into paying entrants is required (who may not be willing to pay the ticket price, which might perhaps be ironically increased due to the cost of the turnstile system). Such exploration of how the turnstile’s software qualities and capabilities would be instrumentally valuable, could elicit value-realisation failure risks, as well as aid the software feature set’s selection and prioritisation. (E.g., to identify the minimal feature set necessary for being able to ‘convert’ zoo guests of the ‘why should I pay when I can get in for free?’ mentality to the ‘will visit and pay’ mentality.) In conclusion, Jackson is correct that, “the eventual goal is the provision and installation of a machine” since “we are concerned with the description of requirements for systems whose construction is primarily a software development task” [Jack97, p.7]. However, in practice, many RE processes do not give sufficient concern with what the stakeholders would actually find valuable, given that the software existing by itself, would not be.

Summary Point #4: Implication for the Thesis

Understanding chains of value derivation from a software capability to its sources of intrinsic value explores assumptions in the value proposition, as well as alternative or complementary requirements, software, or non-software interventions.

2.1.1.2 Means & Ends: Instrumental Value versus Final Value

Instrumental value is a special type of extrinsic value (and the type that has received the majority of attention from philosophers [Zimm10]), that is generated only when value is derived through a means-end relationship. Despite common belief, the opposite of instrumental value is not intrinsic value, but rather final value [Kors83]. That is, instrumental value is that which is generated through being a means, while final value is generated through being an end. Things are only instrumentally good if their ‘causal consequences’ [Zimm10] help achieve “some worthwhile goal beyond itself” [Krau10]. Some philosophers such as Aristotle believe that value can only be

---

Later sections in this chapter describe the more holistic viewpoint of software development in the context of systems engineering [Ieee11b], where software cannot fulfil all of the system’s responsibilities.
instrumental if it ultimately leads to intrinsic value [Aris50]. Those who do not agree with the notion of intrinsic value obviously reject this definition, since, “in order for Y to confer its value on X, when X is conducive to Y [i.e., X is a means to Y], it is certain that Y must have some value to confer, but whether that value is intrinsic or instrumental does not matter as far as X is concerned.” [Bear65, p.5]. As such, and despite the logical arguments for the necessity of intrinsic value (e.g., the previously discussed ‘nothing comes from nothing’ argument), a definition that all should agree on regardless of their stance on the existence of intrinsic value is that: “X has instrumental value means X is conducive to something that has value” [Bear65, p.5].

2.1.1.3 Means & Ends: Common Misconceptions about Instrumental Value

Knowing when a chain of means-ends starting from a software feature has not been adequately explored, (i.e., where it demonstrates extrinsic rather than instrumental value), is important for modelling how the system-to-be will create value for its stakeholders. Zimmerman describes common pitfalls in classifying value as instrumental [Zimm10], which have been adapted here for applicability to the software requirements engineering context:

**Intentionally Good, not Actually Good:** For example, if an information system were to be developed for use in an organisation, the system would not likely be considered successful if, while it could have generated value, it did not (e.g., if its users chose not to use it). Hence, the system would only be considered extrinsically good because of the opportunity provided to the organisation. On the other hand, it could be argued that the un-used system generated instrumental value through the generation of experience and knowledge for the system’s developers — but what it was instrumentally good for matters. A retrospective evaluation of the software project would most likely concern the software’s causal contribution to the outcomes originally desired and paid for by the stakeholders. (Such a distinction between decisions and outcomes has been traced back to as far as Herodotus in 500 BC: “a decision is foolish even if it led to good consequences, if it was unreasonable to expect those consequences given the evidence at hand” [Jayn86].)

**Indicatively Good, not Causally Good:** For example, in the relationship between a heart rate monitoring system and a patient’s health, the heart rate information is indicative of good health (extrinsically good) rather than a means to it (instrumentally good) [Zimm10]. Instead, the heart rate monitor only directly causes improved knowledge of the patient’s health, which itself may be instrumentally good for making the decision to pursue medical intervention.

**Good for All, not for Some (People):** Consider the previous example, but where the patient is a serial killer. The potential for future murders enabled by the medical intervention might outweigh the value generated by saving the killer’s life. Simply put, “one man’s villain is another man’s hero”, and “one man’s rubbish is another man’s treasure” [Cham79, p.598]. Similarly, whether or not a software feature is considered instrumentally good depends on the viewpoint: who is evaluating the goodness? (e.g., profiting executives or an employee made redundant by software). Furthermore, a summative evaluation of goodness encompassing all stakeholder perspectives would require agreement on a normative ethical theory [Kaga98b] (e.g., utilitarianism for its ‘greatest good for the greatest number’ policy).

**Only Positively Good, not Negatively Bad:** For example, a heart rate monitoring system reduces the likelihood of an untreated heart attack (intrinsically bad). Causally contributing a reduction of negative value, is instrumentally good, just as is contributing an increase in positive value.
Overall, this mature and well-established distinction between types of value provides a philosophical underpinning that allows us to reason about how software features can be instruments for creating value for its stakeholders. Unfortunately, the richness, complexity, and subjectivity of these philosophical concepts has often been overlooked or confused with definitions from other fields by practitioners applying them in a business context. For example, Wicks states that “Economists like to talk about intrinsic and instrumental value, but we have known since Aristotle that ultimately all value is instrumental” and provides the example that that “Gold may have intrinsic market value, but it was of no value to Midas” [Wick12]. However, as this section has discussed, philosophers are not of the opinion that all value is instrumental (extrinsic, perhaps), since some value logically has to be ‘final’ [Kors83]. Furthermore, while gold may have intrinsic value according to an economic, numismatic, or financial management definition [BrHo11, p.12], it was not intrinsically valuable for Midas (to use Wicks’ example) according to the philosophical definition; Despite it ultimately being an individual’s subjective distinction to make, gold would not likely pass Moore’s isolation test, and neither does it nor wealth feature in Frankena’s list of intrinsic values [Fran73].

2.1.2 Cause & Effect

As a means to an end, a software requirement is good because of its causal consequences. Requirements are proposed causes, prescribed to bring about some desired effects in the problem domain (cf., Jackson’s ‘two worlds’ view of requirements [Jack95a]). On the other hand, requirements are also proposed effects that some system or system component will be responsible for (causally, not consequentially [Somm07a]). This section provides a conceptual understanding of causality, to provide a theoretical underpinning for the modelling of how software requirements cause benefit to be realised, through chains of means (planned causes) and ends (desired effects).

All of our knowledge “about the world” (i.e., Hume’s a posteriori ‘Matters of Fact’) is founded on the relation of causes and their effects experienced in constant conjunction [Hume48]. This is in contrast to knowledge attainable through logical reasoning alone (i.e., Hume’s a priori ‘Relation of Ideas’), e.g., mathematics [Reyg03], in which “you can see that it is true just lying on your couch” [Whit09, p.36]. Hence, while we can be certain in our knowledge (of ideas) that ‘a GUI [Graphical User Interface] is a UI [User Interface]’ we can only have some degree of confidence in our knowledge (about the world) that ‘a GUI is an effective UI’. In this case, every time we experience the cause (e.g., using a GUI) leading to the effect (e.g., satisfactory completion of a user interaction goal), the degree of confidence in the causal proposition increases. As such, when planning to affect something in the world (as is the case in ‘engineering’: “to bring something about” [Oxfo13]), robust knowledge of the causes that bring about the desired effect(s) (i.e., to create, produce, affect, or alter outcomes [Suss91, p.637]) is valuable when enumerating and evaluating the possible means to it. Otherwise, the planned requirements are merely hypothesis waiting to be tested by implementation.

Summary Point #5: Implication for the Thesis
While ‘means to an end’ is a popular idiom, the philosophy behind ‘instrumental value’ is not so often understood. It is often confused with extrinsic value, contributory value, indicative value, intentional value, positive value, or economic value. Avoiding the common pitfalls in this section would improve the descriptiveness and trustworthiness of value models.

Summary Point #6: Implication for the Thesis
While the correctness of software requirements can sometimes be validated through logical reasoning alone (a priori), validating the benefit of a prospective software requirement requires a historic knowledgebase of previous cause and effect (a posteriori). Hence, there is strong incentive for making models of value explicit, i.e., documenting for later re-use.
2.1.2.1 Causality: The Concept

Examples of causal claims range from being colloquial ‘old wives’ tales’ (‘an apple a day keeps the doctor away’, etc.) to being scientifically validated (e.g., ‘Advanced Encryption Standard (AES) encryption prevents data theft’ [DaRi03]). Interestingly, despite the ubiquity of causal propositions, they rarely refer to a single, shared conceptual understanding of causation [PaWe01]. For example, the former claim could plausibly be interpreted as ‘all possible instances of data theft will be prevented with AES encryption’, or ‘at least one instance of data theft may be prevented by AES encryption’.

As a result of this ambiguity, Kramer et al. propose to classify causal propositions [KrLa92] by their:

- **Timeframe**: whether they are *retrodictive* (‘X did cause Y’), *potential* (‘X can cause Y’), or *predictive* (‘X will cause Y’), and;
- **Locus**: whether they apply to *populations* (‘X can cause Y in any set of circumstances’), or to *individuals* (‘X can cause Y in a specific set of circumstances’).

In practice, retrodictive causal propositions about populations are often translated into predictive causal propositions for individuals. This is perhaps most evident in healthcare, where, for example, a medical practitioner might claim that “giving up smoking will actually lower individual’s probability of developing lung cancer” [PaWe01]. Despite the use of certain-sounding terms such as ‘will actually’, such translations are inherently uncertain due to differences in circumstances from the population to the individual. Nevertheless, all problem solvers (including requirements engineers [RoRo12a]) must perform similar translations, since there is no alternative source of knowledge ‘about the world’. As an example, van Lamsweerde’s RE process for deriving an abstract solution specification from a stakeholder’s need [Lams09b] can be summarised in terms of Kramer et al.’s **timeframe** and **locus** causal proposition classifications:

1. **Retrodictive causality** knowledge is used when enumerating the alternative designs that *did* solve the same problem in previous similar projects;
2. **Potential causality** is postulated when selecting the designs that *could* solve the problem in the particular project, taking project specific context into account;
3. **Predictive causality** is estimated when evaluating each alternative design for the degree to which it *will* (or rather, *would* [Ster00, p.139]) be the optimal design.

<table>
<thead>
<tr>
<th>Summary Point #7: Implication for the Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>When evaluating confidence in the predictive causal proposition that implementing software requirement X will lead to value Y, the applicability of retrodictive causal knowledge from previous projects to the current project's context must be considered.</td>
</tr>
</tbody>
</table>

Aside from the timeframes and locus of causal claims, there are numerous interpretations on what it means for X to cause Y. Theories of causation have been constructed for two different but interdependent purposes: for the development of theory in order to better explain the world, and for practice in order to intervene in some problematic scenario. A review on causation within the philosophical (covering ‘theory’) and epidemiological literature (covering ‘practice’⁶) revealed five key definitions of causality [PaWe01]: *Production*, *Necessary*, *Sufficient*, *Probabilistic*, and *Counterfactual*. The remainder of this subsection will describe these definitions and their applicability to describing how a software requirement could cause some beneficial effect(s).

---

⁶ Epidemiological practice concerns the problem of reducing morbidity by building causal models of disease.
Production, Necessary, and Sufficient causes

Production causes are conditions that play essential parts in creating or producing the occurrence of the effect [Suss91]. This is the definition closest to the layman’s definition, and is the least useful of the five, since it is defined only in terms of the undefined concepts of creation and production.

Necessary causes are conditions without which the effect cannot occur [Steh85], i.e., the effect cannot be true unless the cause is true. For example, ‘Having Electricity’ is necessary for ‘Laptop is Powered On’, and through contraposition, ‘Laptop is Powered On’ implies ‘Having Electricity’. However, it does not cover all types of causality, since a cause may be able to contribute to an effect without it being necessary. For example, ‘Laptop is Powered On’ does not necessitate ‘Having Mains Electricity’, since it can be substituted with ‘Having Battery Electricity’. Following on from this example is the disadvantage that claims of necessity are highly sensitive to ambiguity. For example, while not necessary for the whole population of cases, ‘Having Mains Electricity’ for an individual case may be necessary if the laptop’s battery is discharged. Overall, despite that some causes may be necessary for their effects, “necessary causation is inadequate as a definition of causation” [PaWe01].

While the dictionary definition of ‘requirement’ as “something demanded or imposed as an obligation” [Coll14] implies that a requirement is a necessary cause for effects desired by stakeholders, in terms of causation, the majority of requirements would actually be considered to be probabilistic sufficient-component counterfactual causes (which is introduced later in this section).

Sufficient causes guarantee that the effect will occur, i.e., the effect must be true if the cause is true [PaWe01]. Descriptions of sufficient causes tend to be more extensive than of necessary causes, due to the thoroughness of argument entailed by guaranteeing truth as opposed to falsehood. For example, while numerous resources are independently necessary for human life (air, water, food, etc.), a multitude of resources are necessary in combination to be sufficient for it. Hence, specifying sufficient causes for an effect requires exhaustively negating non-obvious exceptions that could break the guarantee. For example, ‘Providing a GUI’ may be considered sufficient for ‘User-Software Interaction’, but no such interaction could occur if the ‘User is Blind’, the ‘Monitor is Off’, and so on ad infinitum. Consequently, many claims of sufficiency made in practice are technically invalid, since “there does not seem to be any practical limit to the number of necessary conditions” [Swar97] often required to guarantee an effect. (An amusing illustration of this is the ‘Open-Source Wish Project’ that attempts a fool-proof wording for a safe ‘Wish for Immortality’. Despite its comprehensiveness, it can easily be re-interpreted to yield a wish undesirable to the wisher [Yudk11].) As a result of this non-trivial obstacle in specifying sufficient causes, along with the prevailing need for admitting causes which may not be necessary, Rothman proposed the sufficient-component cause model [Roth76] as a derivation of Mill’s ‘constellation of causes’ [Mill43].

Sufficient-component causes

A sufficient-component cause is defined as “a minimum set of factors and circumstances [component causes] that, if present in a given individual, will produce the disease [effect]” [Roth76]. Component causes are independently contributory and insufficient for the production of an effect, but are jointly sufficient. According to this view, causality can be represented as a ‘pie chart’ of component causes (slices). Each pie represents one pathway for the production of an effect, and there can be many pathways (pies) per effect. Figure 2.3 demonstrates this view of causality for a desirable software engineering outcome as the effect (rather than its typical context of an undesirable disease).
The major contribution of the sufficient-component cause model is that it is no longer necessary to exhaustively enumerate the set of component causes for an effect, since it embraces the incompleteness of any such set (represented by slice [U] in Figure 3). These ‘countless’ other hidden component causes (or ‘effect modifiers’) [PaWe01] may either be unknown or omitted. Hence, we can now claim that ‘Providing a GUI’ is a component cause for the effect ‘User-Software Interaction’, without exhaustively specifying or even knowing the remainder of the sufficient cause (e.g., ‘Monitor not Off’, ‘User is Literate’..) which would guarantee the effect. In other words, ‘Providing a GUI’ is an Insufficient and Non-redundant part of an Unnecessary but Sufficient (‘INUS’ [Mack74]) condition (i.e., causal pathway) for ‘User-Software Interaction’. The reader may be wondering how this is advantageous for Requirements Engineering: is it good to leave ‘effect guaranteeing’ causes unspecified assumptions? The assumption of incompleteness is considered acceptable for epidemiology where the goal is to prevent diseases from occurring, since preventing any one of the specified component causes would be enough to prevent the pathway to the effect.

However, Requirements Engineering has the much more ambitious goal of eliciting a complete set of causes to guarantee an effect, i.e., to satisfy stakeholder goals [Lams09a]. (Indeed, a major cause of IT project failure is not identifying the “necessary means for benefits realisation” [LiPe03].) In doing so, there is a fine line between exhaustively specifying perhaps trivial component causes as requirements, and producing a rigorous requirements specification that identifies assumptions which otherwise so frequently cause system failure [Ukhe03].

On balance, the ability to specify component causes rather than exhaustive sufficient causes is useful in Requirements Engineering, since it is interested in causality not to build theories of the world, but to solve practical problems. As such, any potential component causes which cannot be satisfied by the system-to-be’s agents (software/hardware/humans) are not valid requirements [Lams09a, p.317]. Hence, requirements such as ‘The user must not have sudden vision loss after completing the login form to be able to view their account details’ can safely be left out of specifications in Requirements Engineering, since the system-to-be’s agents cannot cooperate to adequately enforce it. This is despite that ‘not having sudden vision loss’ is a valid component cause for ‘being able to view account details’. Therefore, the perhaps trivial requirement that ‘The computer must have electricity at all times to run the software’ would be considered a valid requirement in most systems, since its agents would likely be able to cooperate to enforce it, e.g., through the purchase of battery backup power supplies. Nevertheless, despite its correctness, the triviality of a requirement is
ultimately context dependent. E.g., for a security surveillance system or medical device, power loss could lead to catastrophic failure of the system’s goals, and so that requirement would not be trivial.

**Summary Point #8: Implication for the Thesis**

A software capability prescribed by a requirement is only part of a causal pathway for the desired effect, which itself is instrumentally good for another effect(s). The other component causes for the effect need to be planned (achieved/avoided).

The final noteworthy advantage of the sufficient-component cause model is that it clarifies and disambiguates the definition of a necessary cause in view of the pie chart representation of causality: A necessary cause is a causal component present in every possible causal pathway (pie) that produces the effect. Hence, if a necessary cause for an effect is prevented, the effect cannot occur in any set of circumstance, i.e., any pathway. Therefore if a new causal pathway for the effect ‘User-Computer Interaction’ (with reference to Figure 2.3) were found which did not require slice [D] (‘User not Blind’), e.g., perhaps involving ‘Providing an Audio User Interface’, then it could no longer be considered necessary for the effect, despite that it is required for causal pathways #1 and #2.

**Summary Point #9: Implication for the Thesis**

Claims about a software capability being ‘necessary’ to create some value for a stakeholder are likely to be highly ambiguous about the context for ‘necessity’, unless they are made under the definition of the ‘sufficient-component’ model of causality.

As for disadvantages, the sufficient-component definition of causation is considered by some to be “ontologically unwieldy” due to its forced assumption of unknown causal components, and acceptance of absent evidence for an effect’s cause is deemed by some to be ‘unscientific’ [PaWe01].

Because a sufficient cause is fully sufficient for its effect, it is not straightforward to describe the impact that adjusting a cause (e.g., varying a system availability requirement for an e-commerce website from 99.8% to 99.9% monthly uptime) would have on its effects (e.g., a reduction in lost sales). Changes in the quantity of a component cause can be represented in a new causal pathway (a new pie chart) [Roth76], but this captures dose-response relations as a series of discrete steps, rather than as a continuum [PaWe01]. Furthermore, in this definition of causality, it is the causal pathway rather than the component parts that are said to cause the effect. As such, it is not possible to attribute more causal responsibility to one particular causal component, e.g., the software’s provision of a GUI ([A] in Figure 3) than the user not being blind ([D] in Figure 3).

Furthermore, the sufficient-component cause definition “retains an assumption of scientific determinism that often goes unacknowledged” [PaWe01], whereby the same causal pathway always brings about the same effect with no room for variation or chance — i.e., the effect is wholly determined by those circumstances. Proponents argue that chance can be viewed as “the result of deterministic events that are beyond the current limits of knowledge or observability”, in the same way that a coin flip can now be “determined completely by the application of physical laws and a sufficient description of the starting conditions” [RoGL08, p.17]. Others argue that determinism is a restrictive and non-natural simplification that misrepresents the real world [Fish34], and that it is practically unable to describe many cases where stochastic models are essential, e.g., chaos theory.

As a result of these criticisms, the most accepted definition of causation is probabilistic causation [Mell95], which is inclusive of all of the mentioned concepts from the previous definitions of causality.

**Probabilistic causes and counterfactual causes**

**Probabilistic causes** are said to increase the likelihood (probability) of their effect occurring [Supp70], e.g., ‘AES encryption will increase the probability that personal data will not be stolen’.
Rather than ‘guarantee’ that personal data will not be stolen (sufficient cause), or claim that without ‘AES encryption’ personal data will be stolen (necessary cause), the probabilistic cause definition admits that any cause is just one contributing factor in a random (stochastic) process that brings about an effect. Hence, there is a great deal of similarity between the sufficient-component model of causation and the probabilistic definition. Indeed, Rothman and Greenland propose that the sufficient-component pie chart model of causation could be adapted to be probabilistic [RoGL08]: each of the components (pie slices) would contribute together to the probability of the effect, rather than being sufficient for it. Crucially, rather than each component being equally important (note the equal pie slice sizes in Figure 2.3), each component cause would have a degree of influence, quantified in terms of probability. That is, the increase in probability for the effect when the component cause is present, as compared to when it is absent. If a component cause is missing (i.e., prevented or avoided), then the probability of the effect occurring is simply decreased. Hence, the probabilistic definition is inclusive of the previous definitions for causality, since a probabilistic cause can be either necessary (the probability of the effect equals zero if missing), sufficient (the probability of the effect equals one if present), or a partly sufficient component-cause (the probability of the effect increases if present). The same cannot be said for the necessary and sufficient (deterministic) definitions, where the influence of a cause is ‘all or nothing’.

According to probabilistic causation, an effect’s occurrence or non-occurrence might be “in part, a matter of chance” [PaWe01]. For example, it is incorrect to claim that ‘AES encryption’ of data is causally sufficient for preventing theft (decryption), since a brute force attack could by chance rapidly predict the correct key, regardless of the strength of the encryption scheme [Bokl09]. Most agree that the complexity of causal variation due to uncertainty is best described with the probabilistic definition of causality, since “probability is perfect” in that it “is uniquely appropriate for the representation and quantification of all forms of uncertainty [i.e., aleatoric and epistemic]” [OhOa04].

While the previously discussed definitions of causality provide only the often unsatisfactory knowledge that “x is part of a constellation of causes sufficient for y” [Mill43], probabilistic descriptions communicate knowledge about the strength (i.e., degree, extent, amount) of a causes’ contribution to an effect. Given that knowledge of x’s relative size among y’s causes is often necessary for practical intervention to affect y [PaWe01, p.910], the probabilistic definition of causality has gained popularity among researchers and practitioners for pragmatic reasons.

Summary Point #10: Implication for the Thesis
Practically useful causal propositions about a software capability’s predicted value are likely to require estimates of causal strength and the likelihood of that strength, so as to be able to understand and evaluate its adequacy for the desired effect.

Finally, the probabilistic definition provides a means to construct models of “dose-response relations in quantitative terms through a continuum of probability values” [PaWe01]. As aforementioned in the criticism of the sufficient-component definition, it is common for a cause to have various possible doses, e.g., {500, 1000,...}mg of paracetamol in the medical context, or {200, 300,...}ms response time in the software context. These different doses can be mapped to the probability for the effect’s likelihood and/or size (either manually plotted, or specified by a parameterised distribution), as exemplified by Figure 2.4 and Figure 2.5.
In order to ‘engineer’ a solution (as opposed to less rigorous means of construction), the effects of each alternative intervention must be estimated, simulated, and evaluated [Wier96, p.44], and so dose-response knowledge such as that represented by Figure 2.4 would undoubtedly be useful for deciding on the adequate level of a requirement’s prescribed target, e.g., a value of $\theta$ in Figure 2.4.

### Summary Point #11: Implication for the Thesis

A quantified requirement (i.e., specifying an upper or lower bound on a variable for the requirement to be considered satisfied) is just one possible ‘dose’ out of many in a dose-response causal relationship. When choosing the prescribed dose, it is important to know the effect of adjusting it, e.g., when ‘relaxing’ the requirement to reduce the cost of the system.

Unfortunately for the simplicity of the required models, many effects that are desired in software engineering also have numerous degrees of an effect’s satisfaction. For example, with reference to Figure 2.5, the amount of ‘Website Load Time Decreased’ as a result of various values of ‘Website Cache Lifetime’ can vary in milliseconds, which is contrary to Boolean events (e.g., ‘Website Visitor Abandons Interaction’). As such, representing the probabilistic causal relationship between two numerical scales such as for ‘Website Load Time’ and ‘Website Cache Lifetime’, requires three dimensions (Figure 2.5) rather than two (Figure 2.4).

The ‘simulation’ of a cause’s effects (as mentioned two paragraphs ago), and hence the construction of the numerical relationships shown by Figure 2.4 and Figure 2.5, can be performed with different degrees of rigour, each acceptable in different circumstances. Simulation methods range from non-probabilistic rules-of-thumb (e.g., that one gram of uranium gives on megawatt day of energy [Wier96, p.44]) to uncertainty reducing computations (e.g., Monte Carlo experiments [Thom12]). The knowledge required for simulation can be ascertained from previous experience with similar systems, from expert estimates, or from experiments with prototypes, amongst other ways [LeLa04, p.9]. This knowledge should not be left unvalidated after the project has completed, otherwise errors may propagate to future projects. Overall, simplistic but validated rules of thumb, are more likely to be useful than more complex but unvalidated models.

---

7 Boolean events might alternatively be modelled as a threshold on a numerical scale, e.g., the Boolean ‘Visitor Bounced’ event may be considered to be ‘<2’ webpages viewed before the user exits the website.
Overall, the probabilistic definition of causality provides the most inclusive representation of reality. However, there is an unaddressed but crucial aspect of causality: distinguishing between causal and non-causal association. Hence ‘Counterfactual cause’ is the final noteworthy definition of causality.

**Counterfactual causes** make a difference in the outcome (or its probability of occurrence) when it is present, compared with when it is absent, while all else is held constant. The counterfactual definition of causality complements the others (indeed, it is insufficient by itself), and resolves the problem of defining an associative rather than a causal link. To propose a counterfactual cause is to “draw a contrast between one outcome given certain conditions and another outcome given alternative conditions” [PaWe01]. Applied to the probabilistic definition of causality, a counterfactual cause is defined as: “C causes E if the probability of E given C is greater than the probability of E given the absence of C, while all else is held constant” [Cart89].

The ‘all else is held constant [i.e., equal/unchanging]’ clause is known as the ‘ceteris paribus’ assumption [Lind02, p.14], and is required since a myriad of possible changes in the state of the world (i.e., ‘all else not being constant’) may affect the causal proposition, thereby vastly complicating the modelling of it. For example, if business process X is to be automated by software capability C, then precise claims about the value of C would become nullified if a new business process Y was introduced that made X redundant, i.e., if the state of the world was not held constant. (If such a scenario is likely, then that risk should be made explicit.)

Finally, the ‘absence of [cause] C’ in [Cart89]’s definition of a counterfactual cause, is not always the most suitable baseline for modelling the value of ‘C’. For example, if it was decided that the requirement would not be implemented (i.e., cause ‘C’ would not occur), but the expected benefit (effect ‘E’) is still desired, then the stakeholders might implement an alternative way to achieve ‘E’. Hence, there are two ‘modes’ of modelling the value of a software requirement, where its value is:

- compared to the ‘as is’ current state of the world (without the requirement) by default, or,
- compared to the ‘as would likely otherwise be’ state of the world, when it is known that in the absence of a requirement, an alternative means to achieving the desired end would be implemented, such that the ‘as is’ state of the world would not be held constant.

**Summary Point #13: Implication for the Thesis**

Causal propositions about the amount of value a software capability would create are ‘counterfactual’, given that we wish to model the benefits of the decision to implement a software requirement, compared to the decision not to do so. Hence, the causal proposition for the requirement’s value must be defined with respect to a ‘reference condition’ [RoGL08, p.8]. This may be either the ‘as-is’ situation with the requirement not satisfied, or an ‘as would be’ situation if an alternative would be implemented.

### 2.1.2.2 Causality: Modelling Perspectives

It is not possible to model **all** of the details of a real world sociotechnical system (i.e. comprised of machines and humans with free will) with 100% fidelity to the actual system [Hitc03, p.135]. Even if, for every event, we could learn all of the conditions that could cause only that event (assuming causal determinism), it would not likely be a worthwhile pursuit in the context of software/systems engineering, given the large number of possible conditions to elicit even for simple events. Hence,
the term ‘modelling’ in this thesis relates to the activity of creating a mutable [Roth89] simplification [BeBG05] (i.e., abstraction) that allows one to reason about the subject, by focusing on the relevant details while ignoring the irrelevant [BoBo07].

Modelling and analysing a system’s behaviour through the abstraction of cause and effect has a long tradition of success (e.g., Newton’s discovery of the three laws of motion [Hitc03, p.136]), since “often the details of how some change occurs are irrelevant” [FoGe83, p.6]. As such, Sterman summarises that modelling a system behaviour’s causality is “the art of simplification” [Ster00, p.166].

There are essentially three competing perspectives for modelling cause and effect: Disjointed, Linear Control, and Causal-loop Nonlinear Feedback [Hitc03, p.137], as is illustrated by Figure 2.6.

![Figure 2.6: Viewpoints on modelling cause and effect, recreated from [Hitc03]](image)

Politicians (and past scientists [Evan93]) tend(ed) to take a disjointed view of the world, where cause and effect supposedly exist in discrete pairs of ‘one cause, one effect’ [RoGL08]. For example, ‘urban decay is caused by lack of government investment’, or ‘online data security breaches are caused by low data encryption strength’. This disjointed viewpoint allows singular and convenient ‘silver bullet’ solutions to be proposed, but it does not recognise that there tends to be multiple component causes per effect (strung together in a ‘web’ or ‘chain’ of causation [RoGL08]). As such, this view of cause-effect has been described as “the antithesis of systems thinking” [Hitc03, p.137], and is considered by many to be a “simplistic mis-belief” [RoGL08], causing ineffective solutions.

Project managers tend to take a linear control view of cause and effect, since it allows activities to be strung together, where the output of an activity becomes the input of the next, as in GANTT and PERT charts [Faza59]. Despite acknowledging many causes for one effect, it ignores feedback (loops).

Closer to the real world is the causal loop nonlinear feedback view of cause and effect, where a cause results in an effect which can feed back to interact with the cause. The difference to the linear view of cause and effect is that the ‘first cause’ can be affected by another cause, and hence it can simultaneously be both a cause and an effect. However, this is only applicable where events are specified such that they are repeatable (and hence at least somewhat more ambiguous). E.g., the cause ‘lower price’ can occur many times, while ‘lower price from £400 to £389 on the 15th June 2006’ cannot. Consequently, when modelling the benefit of a software requirement, it is important to be clear about the event (i.e., the cause) that is being modelled, as either (or a combination of):
i. the development or procurement of the requirement;
ii. a single usage of the software capability resulting from the requirement;
iii. an aggregation of many usages of the resulting software capability over some defined timespan.

An implementation, development, or procurement of a software requirement in a project (i) cannot feedback to affect that specific (and now historic) implementation, because once a requirement is correctly implemented, the same requirement would not be developed again within that same project. On the other hand, the usage of an implemented software requirement (ii) can affect its future usage and hence (ii) is viable for causal loop feedback modelling, e.g., the use of a print function decreases ink levels, increasing required spending on ink, decreasing the use of the print function, thereby decreasing the occurrence of the print function’s beneficial effects. However, when assessing the value of a software capability (in scenarios such as deciding among alternative requirements), it is nonsensical in most cases to consider and compare only the value gained by just one usage of it. Rather, the value gained over a timespan of many usages (such as over a two year period of usage) is required to provide a frame of reference that adequately represents the variation of a software capabilities’ usage over time (i.e., due to aleatoric uncertainty). The optimal timespan would vary per instance, but, for example, in financial estimation in software projects, both Cohn & Bills suggest a two year period for the balance between ‘not looking far enough ahead’ and ‘guessing at the distant future’ [Cohn05, p.89]. Hence, if the requirement as a cause is modelled as an aggregation of usages (event type iii), then the modelling complexity introduced by feedback loops (turning acyclic graphs of cause and effect into cyclic graphs) can be avoided, while still considering the important aspects of feedback loops. (Thus, in the context of Figure 2.6, requirements could be modelled with the linear control view, and effects of feedback included in estimates on the number of times the software capability is likely to be used in the timespan.)

Summary Point #14: Implication for the Thesis
The complexity of value models can be reduced (by avoiding loops) if the causal event representing the requirement encapsulates a time period of the implemented requirement’s usage, rather than just its development, procurement, or one usage of it. Hence, the ‘value of a requirement’ should be read as ‘the value of implementing, and using it many times’.

As Figure 2.6’s linear and non-linear viewpoints show, every event has a preceding cause, “which in turn is an effect of some still earlier cause” [Ster00, p.10]. Hence, chains of cause and effect can be constructed until an event’s ‘root cause’ is reached. E.g., ‘inventory is high’ because ‘sales are low’, because ‘competitors lowered their price’, and so on. This chain of reasoning can be indefinitely extended until some ‘First Cause’ is reached (as is the premise of Aristotle’s cosmological argument for the existence of God [Aris00, pp.4–6]), and as is also the logical argument for the existence of ‘pure’ intrinsic value (as aforementioned in Section 2.1.1). Regardless of this, practically speaking, the stakeholders’ interest in extending a chain of cause-effect events will most likely diminish well before matters of religion are reached. Indeed, the ‘5 Why’s’ Lean Manufacturing method for root cause analysis considers five events (causes) to be sufficient extension of a causal chain [Ohno88].

2.1.2.3 Causality: Mechanistic Descriptions

This thesis is interested in describing a software requirement’s mechanism for creating benefit, where a mechanism is the continuous or discrete process that is responsible for causal change [FoGe83, p.2]. While the validity of a causal proposition is unaffected by whether it is mechanistic or merely explains the ‘phenomenal primitives’ [Dise83], the former is considered to be superior for its ability to aide comprehension and reasoning about cause and effect. Interestingly, both extreme novices and advanced experts in a domain tend to show non-mechanistic reasoning about causal changes in
quantities [FoGe83, p.9]; experts consider the mechanics to be trivial, while novices do not possess the domain knowledge.

Non-mechanistic causal reasoning about software requirements would not positively address the problems motivating this thesis (e.g., that un-used software functionality is developed). Rather, it is considered to be analogous to a tired parent answering their inquisitive child’s “why?” questions with ‘it just is’. Instead, if faulty causal propositions (i.e., requirements that would not actually be beneficial) are to be identified, thus preventing wasteful implementation, it would be through the explication, communication, and refutation of the software’s causal mechanism for creating benefit.

For example, it is unlikely that a stakeholder would refute the non-mechanistic causal proposition that ‘having a copy and paste function’ (cause) would ‘improve usability’ (effect), due to its correctness on average over all instances of software. However, a mechanistic causal proposition that it would ‘reduce manual re-entry of identical data within a user interface’ (effect), which would cause a ‘perceived improvement to usability’ (effect), enables a stakeholder to query its applicability to the individual software project’s context. E.g., perhaps manual re-entry of identical data does not, nor would occur, and so requiring a ‘copy and paste’ functional requirement would be wasteful.

**Summary Point #15: Implication for the Thesis**

Causal propositions about a software feature’s expected benefits must be mechanistic to be easily refutable (hence useful).

2.1.2.4 Causality: Common Misconceptions and Errors

Many planned means to ends do not bring about their ends due to an incorrect understanding of the cause and effect relationships for the desired end. Researchers report numerous reasons for this:

**Unvalidated Theories:** Modelling that a software capability will cause some benefit will not by itself make it so, in the same way that writing a medical textbook claiming that bloodletting causes epilepsy to be cured did not make it so [Woot07]. Rather, only the elucidation, verification and validation of the benefit causing mechanism can help expose faulty beliefs or determine confidence. Use of evaluated causal propositions is important. For example, Kohavi has found that “80% of the time, you are wrong about what a customer wants”, and so “data trumps intuition” [Koha13, p.21].

**Summary Point #16: Implication for the Thesis**

Modelling a causal proposition that requirement X will lead to benefit Y will not make it so. It must be communicated, reasoned about, and validated in order to change, or build confidence in the requirements. I.e., modelling is itself not intrinsically valuable.

**Lossy Communication:** Many software engineering tasks in industry are “carried out without question, which are no longer appropriate for today” [Nola99, p.212]. Numerous vivid stories [HaPr94, Mats97, Nola99, p.212] have been disseminated about the adverse effects of miscommunicating causal propositions, e.g., communicating only A→C in chain A→B→C. They can all be summarised by Matson’s claim that policies, precedents and processes “often outlive the particular industry context that created them” [Mats97], i.e. where B in A→B→C is no longer relevant. Non-mechanistic propositions are partly to blame for this, since they hide how changes in context could nullify them.

**Weak Analogy:** “Van Gogh was poor and misunderstood in his lifetime, yet he is now recognised as a great artist; I am poor and misunderstood, so I too will eventually be recognised as a great artist.” [Warb07, p.142]. This fallacy can be easily applied to Requirements Engineering, where stakeholders may prescribe requirements because similar successful projects had them. Indeed, an ex-Microsoft employee claims that “useless Internet shopping buttons started cropping up” on Microsoft keyboards due to their belief that ‘Internet components’ cause successful products [Spol00].
Bias and Irrationality: Knowledge is accumulated via the senses of individuals, which are subject to faults such as illusion [HePB84], inaccuracy [HuHe95], bias [TvKa74], or ‘Fear, Uncertainty and Doubt’ (FUD) [HaMc98, p.224]. Hence, a group’s understanding of an effect’s causes, or the strength of a cause (and hence the optimal means for an end), may be diverse or irrational. For example, stakeholders exposed to, and hence most knowledgeable about one sub-process in a problematic process tend to over-exaggerate its causal importance [Ster00, p.18]. Alternatively, a stakeholder might request inclusion of software capabilities or qualities due to their fear of the consequences of the software project’s failure outweighing their desire to be economical. This reasoning is embodied by the infamous “No one ever got fired for buying IBM [expensive] equipment” quote [Anon78, p.58].

Correlation not Causation: “Up to 1889, men of science had thought only in terms of causation, in future they were to admit … correlation” [Pear30]. Still, the two are often confused because “empirically observed covariation [correlation] is a necessary but not sufficient condition for causality” [Tuft03, p.4]. As such, propositions about value being caused by a software capability should be examined for other viable causes, in order to avoid specious reasoning. Significant effort is required to find “the genuine causal needles in a huge haystack of correlations”, often requiring controlled and repeatable experiments, statistical inference, scepticism, peer review, and so on [Ster00, p.142]. The ‘Hill criteria’ are well accepted for evaluating whether a correlation is causal [Hill65]:

1. **Strength**: weak associations aren’t not causal, but strong associations are more likely to be;
2. **Consistency**: are similar findings observed in different places, times, and by different people?
3. **Specificity**: the fewer the other likely explanations for the effect, the stronger the proposition;
4. **Temporality**: “which is the cart and which is the horse?”: the cause must precede the effect;
5. **Dose-response relationship**: the effect size should be proportional to the size of the cause;
6. **Plausibility**: it is easier to accept causality when there is rational and theoretical basis;
7. **Coherence**: generally, the causal proposition should not contradict other sound knowledge;
8. **Experiment**: by which, “the strongest support for the causation hypothesis” may be revealed;
9. **Analogy**: evidence of causation in slightly different causes/effects can strengthen the proposition.

Unfortunately, such efforts are rarely perceived to be affordable in a typical software project (politically and financially), instead defaulting to merely observing whether the software and the benefit co-exist. However, the accuracy of future value predictions depends on validated causality.

<table>
<thead>
<tr>
<th>Summary Point #17: Implication for the Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A stakeholder’s perception about the degree to which a software capability would be beneficial is likely to be somewhat distorted. This is not a reason to avoid making such perceptions explicit, but rather why they should be communicated. Then, the soundness of the (otherwise implicit) causal proposition can be evaluated, e.g., using the Hill criteria for causation.</td>
</tr>
</tbody>
</table>

While many of the pitfalls described in the above section originate from the application of past knowledge to future contexts, the concern is the quality of the interpretation and application of previous knowledge, rather than whether or not it is used, and so Summary Point #68 is not conflicted.

2.1.3 Benefit, Cost, & Value

A software requirement is expected to be beneficial, just as any ‘end’ in a ‘means to an end’ is expected to be (Cicero’s principle of selection in Section 2.1.1). That is to say that a positive amount

---

8 Summary Point #6 stipulates that a knowledgebase (of uncharacterised quality and completeness) of ‘software capability→value’ pairs is required to be able to propose requirements with any confidence.
of benefit should be provided after the costs of implementing it and using the resulting capability have been considered. This calculated net benefit is often referred to as value [Busi14a].

2.1.3.1 Benefit

A benefit is considered to be “something that is advantageous or good” [Dict14] about “an outcome of change which is perceived as positive by a stakeholder [or group]” [Brad06], that could be realised “through the delivery of objectives” [Turn08]. In the software project context, Boness notes that benefit is generated through ‘the outcome of the operation of the output of a project’ [Bone11, p.9].

<table>
<thead>
<tr>
<th>Summary Point #18: Implication for the Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>It would be logical to assume that an objective is expected to be beneficial, and hence that objectives are synonymous with expected benefits. However, in the context of pair of objectives (A,B) where A contributes to B, the benefit refers to B.</td>
</tr>
</tbody>
</table>

Benefits of software are most frequently classified as being tangible or intangible [HaRo94], where:

1. **Tangible benefits** are measurable, quantifiable and objective [KaAK01];
2. **Intangible benefits** are immeasurable, qualitative and subjective [MuSi01].

Enriching the above classification (in a non-software-specific context), Bradley presents his widely-accepted taxonomy of benefit value types [SaHK08, p.12], shown in Table 2.1.

<table>
<thead>
<tr>
<th>Table 2.1: Benefit Value Types, according to Bradley [Brad06] (Benefits Management Literature)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benefit Value Type</strong></td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>Definite (Tangible)</td>
</tr>
<tr>
<td>Expected (Tangible)</td>
</tr>
<tr>
<td>Anticipated (Tangible)</td>
</tr>
<tr>
<td>Intangible (‘soft’)</td>
</tr>
</tbody>
</table>

Perhaps the most important distinction in Table 2.1 is that of ‘Expected’ benefits versus ‘Anticipated’ benefits, which are more complicated to estimate the value of, due to more uncertain causal mechanisms. For example, in Market Driven RE [KDRN07], a requirement might be anticipated to be beneficial for increasing sales of the software product, but the causal mechanisms for increasing sales are not deterministic, nor well known, and are dependent on factors outside of the project’s control.

<table>
<thead>
<tr>
<th>Summary Point #19: Implication for the Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>What will be benefited by an anticipated benefit will be far easier to model than how much of a benefit is anticipated. While this in no way precludes quantitative models of anticipated benefits, the degree of uncertainty in the model will be higher.</td>
</tr>
</tbody>
</table>

While Table 2.1 is, as aforementioned, widely accepted in the benefits management literature [SaHK08, p.12], it does not include the concept of ‘Unexpected’ (i.e., unplanned or emergent) benefits, which often occur as a consequence of the intervention [AsDo03]. In retrospective cost-benefit analysis these unexpected benefits would likely be included, but by definition, it is not possible to include them in the requirements engineering process — given their ‘unexpected’ nature. Additionally, the notion of ‘Indirect’ and ‘Direct’ benefits is not considered, which separates those which are “by-products or side-effects of the project” from those which are “closely related to the execution of the
As a final remark, Table 2.1’s ‘Definite’ benefit type appears to be just an optimistically named, high-confidence type of ‘Expected’ benefit, since nothing about the future is free from uncertainty.

Interestingly, the literature seems to be in agreement that tangible benefits of software are taken more seriously than intangible benefits, as judged, for example, by whether or not they are typically included in a business case for a project. Bocij et al. claim that on average, around 30% of the stated benefits for IS systems are intangible [BoGH08, p.542]. More pessimistically, Mutschler et al. claim that intangible benefits are often not considered at all [MuBR06, p.40]. Similarly, Eckartz found that 88% of surveyed IS organisations considered financial benefits to be “the most important”, but found that if it were not for the difficulty in defining intangible benefits, they would be valuable in decision making processes [Ecka12, p.78]. Indeed, due to the “seemingly insuperable difficulties” in the evaluation of soft benefits (e.g., the “social merit of certain goods”), “no perceptive economist would deny” that they are often completely ignored [MiQu07, p.14]. Hence, traditional financial evaluations of benefit such as Return On Investment (ROI) or Return On Capital Employed (ROCE) are often inherently incomplete for IT systems [SaHK08, p.54], given that non-financials (increases in innovation, learning, co-ordination, etc.) tend to be pivotal to a project’s success [Ecka12, p.122]. (Nevertheless, they are popular since higher levels of management tend to be responsible for finance.)

The differing reports of inclusion of tangible/intangible benefits in business cases will, in part, be due to different project types within the researchers’ (Bocij et al., Mutschler et al., etc.) sample populations. For example, software projects that focus on improving user satisfaction (e.g., perhaps improving a Graphical User Interface) may claim a higher number of intangible benefits, compared to projects that improve the speed of a manufacturing process (and hence reduce tangible operating costs). Overall, since the costs of achieving intangible benefits are likely to be accounted for [Ecka12, p.121], it seems remiss to not consider their benefits, since otherwise it might appear as if the initiative was, or would not be worthwhile. In the words of Forrester (in the system dynamics modelling context), “To omit such [soft] variables is equivalent to saying that they have zero effect — probably the only value that is known to be wrong!” [Forr61, p.57]. DeMarco takes a stronger line: “As a general rule of thumb, when benefits are not quantified at all, assume there aren’t any” [DeLi03].

Summary Point #20: Implication for the Thesis

Benefits should not be ignored just because they are intangible or uncertain. They can often be reasoned about objectively (e.g., by reasoning about numeric indicators acting as proxies), and are often crucial to the success of IT projects.

2.1.3.2 Costs (& Disbenefits)

Costs are opposite to benefits in that they are negative undesirable value effects [TeWe99, p.17]. Adams and Juleff define costs as the “expenses for goods or services, including money, time and labour” [AdJu03], and continue to distinguish between seven types of cost:

1. **Fixed & Variable Costs**: The relationship between costs and level of production.
2. **Direct & Indirect Costs**: The ability to budget for and assign responsibility for costs.
3. **Lifecycle Costs**: The costs throughout the investment’s entire lifecycle.
4. **Internal & External Costs**: The origin of a cost (inside or outside the organisation).
5. **Acquisition Costs**: The costs of purchasing an asset, after adjustments for discounts, etc.
6. **Historical Costs**: The original monetary value of an economic item (i.e., previous payments).
7. **Opportunity Costs**: The difference between the yield an investment earns and the yield earned if the resources had been spent on the highest yielding alternative investment.
The term ‘disbenefit’ is sometimes used synonymously with cost (e.g., in [Holl12, p.49, SaHK08, p.11]), but its intended usage is more specific, in that it should describe a “disadvantage, something objectionable, something that makes a situation unfavourable, or undesirable effects of an investment” [Ecka12, p.32]. Hence, disbenefits are a subtype of indirect (♯2) lifecycle cost (♯3), such as a user’s cost in time for using a software capability. This is in contrast to direct (♯2) or historical costs (♯6), such as the cost of developing or procuring the requirement.

Disbenefits can be categorised similarly to benefits (i.e., according to Table 2.1’s types of benefit) [Bann08], with one main difference: Unanticipated disbenefits would likely require a remedial reaction, which is contrary to unanticipated benefits, whose main reaction would be to record it for later re-use. Along a different dimension, Remenyi classifies disbenefits as those that are either inherent in the change to be caused by the intervention, or those that are caused by bad practice [Reme95], (i.e., according to Brooks’ division of ‘Essence’ and ‘Accident’ in software engineering [Broo87]). Unavoidable disbenefits should not be ignored when modelling the instrumental value of a software requirement even if they are considered ‘inherent’ or ‘essential’, since they may play a role in managing overly optimistic stakeholder expectations, regardless of their unavoidability.

Overall, software requirements are often accompanied by numerous disbenefits and obstacles to realising their intended benefits. These should be avoided or mitigated [TeWe99, p.17] proactively rather than reactively, due to the ‘cost-to-fix curve’\(^9\). However, ‘traditional’ practice focuses on costs accrued in the software development lifecycle [BrDa99], rather than the ‘whole life’ (or ‘through life’) costs, and hence not on disbenefits. This may be due in part to the consequences of disbenefits often not affecting the software developers or their organisation — the classic ‘What’s in it for me?’ problem — and because considering and estimating disbenefits requires the consideration of uncertain future parameters, and unfortunately, people tend to exhibit ‘uncertainty aversion’ [Epst99].

### Summary Point #21: Implication for the Thesis

Each outcome-of-an-outcome from a software requirement has potential for disbenefit and obstacles to its realisation. These ought to be minimised, or avoided or mitigated proactively rather than reactively, due to consequences of the cost-to-fix curve.

While Adams & Juleff’s taxonomy of costs is comprehensive in breadth, it is not exhaustive in depth. (For example, an on only recently defined software engineering ‘lifecycle cost’ (♯3) is ‘technical debt’ [ToAV13]: When applied to the RE context, it is incurred when a requirement that was considered essential by the stakeholders is not included or weakened in a release, consequently adding to the debt of work to later be ‘paid back’ at greater expense when the stakeholders are sufficiently inconvenienced by the omission.) In recognition of the cost taxonomy being too shallow to be useful for eliciting disbenefits of software capabilities, practitioners are recommended to examine quality taxonomies such as FURPS [GrCa87] or ISO 9126-1 [Iso01] for trade-offs. For example, ‘usability’ costs might be incurred if a requirement’s implementation would increase user-visible complexity, i.e., ‘bloat’ [Mcgr00].

Overall, the primary focus of this thesis is on reducing the opportunity costs (♯7 in Adams & Juleff’s cost types) of a software project’s requirements, since unused or ‘gold-plated’ software capabilities will in most cases have consumed the opportunity (time and money) for developing a more useful capability. More importantly, the opportunity for realising the stakeholder’s expected benefits may be lost. The

---

\(^9\) Boehm’s ‘cost-to-fix’ curve [BoBa01] claimed that it is ~100 times more expensive to fix a software defect in production than if it were fixed during RE (in large projects). Boehm later claimed that in some “small and agile projects” a ~5 times ratio could be found [BoHZ05, p.380], while more recent cost-to-fix calculations by Gartner claim a ~200 times ratio on average across project types and sizes [Gart11]. The ‘cost to fix’ will have high variance between software project characteristics, such as its deployment method (e.g., ‘cloud’ software is cheaper to deploy bug-fixing updates to than ‘install offline’ software).
other costs (#1-6) are considered as inputs to this optimisation exercise, in that reduced ‘acquisition costs’ such as development or licencing costs, would reduce a requirement’s opportunity costs.

2.1.3.3 Value

Notions of value, cost and benefit, desire, goodness, justification, rationale, and so on, are only pertinent to the world because every action taken has consequences and alternatives. Beardsley’s ‘one minute to live’ thought experiment exemplifies this claim: “Let us suppose a child has only a few minutes to live, because the world is about to come to an end, and he is rapturously swinging; and suppose anything else you like, to close off other alternatives. He wants to swing, he is ecstatic about it. Of course we are not going to call him in to do his homework, or worry about his catching cold.” [Bear65, p.14]. However, given that most are not of the opinion that the world is imminently about to come to an end, then acts — in this context, satisfying software requirements — do raise the question of whether they would be justified over alternative uses of the resources (time, effort, etc.) [Bear65, p.16], at least compared to the ever-present ‘do nothing with the resources’ alternative.

**Summary Point #22: Implication for the Thesis**

Exploring the value of requirements is motivated by the existence of opportunity costs; Stakeholders less motivated by cost concerns may be better motivated by pointing out that more valuable software functionalities or qualities could be gained.

The term ‘value’ is often used synonymously with ‘good’. However, not all things considered to be good are valuable (such a software feature of ‘good quality’), and vice versa. More importantly, value has two common interpretations, as is shown by the two links toward the concept in Figure 2.1.

In the first and most common interpretation, the value of something is the “relative worth [to its costs]” of it [Dict14]. Similarly, the Institute of Value Management (IVM) defines value simply as the formula: the “satisfaction of a user’s needs divided by the resources consumed to satisfy those needs” [Inst13], i.e., the cost-benefit ratio (also known as usability-utility, such as in the Technology Acceptance Model [Davi89]). While cost-benefit analysis is commonly adopted best practice for decisions about project selection, it is not for requirements selection [SiNu01]. Finally, despite the ubiquity of ‘cost-benefit analysis’, many practitioners perform inferior ‘cost effectiveness’ analysis [MiQu07, p.10] due to fixation on cost reduction, preference for discussion of tangible entities, and limited understanding of the true benefits. For example, after conducting cost-benefit analysis for software project feasibility studies, corporate executives have been found to often have “difficulties in identifying the exact strategic goal(s) toward which the technological system will contribute”, instead possessing only an abstract understanding [SiWo09].

In the second interpretation originating from Ethics, a value is “any object or quality desirable as a means or as an end in itself” [Dict14]. In other words, it is the object of evaluation in the first interpretation (as is visualised in Figure 2.1). Many values are often desired in combination, sometimes in a state of conflict, e.g., most software users value ‘Security’ in addition to ‘Usability’. A list of a stakeholder’s values can grow unmanageably large, and so Howard advises to only consider ‘Direct Values’ as comparison criteria in decisions among alternatives [Howa07, p.33]. (Howard’s ‘Direct’ and ‘Indirect’ value distinction seems highly similar to the distinction between whether or not a value is instrumentally valuable for another. Hence it will not continue in this thesis.) For example, while an e-commerce business owner might claim to value both ‘Usability’ and ‘Sales’, the former is only considered important because of the latter, and so alternative options should be evaluated for their potential improvement to just ‘Sales’, rather than ‘Sales’ and ‘Usability’. While this guidance may reduce complexity in that context, it is not applicable in this
thesis’ context, since it would prevent mechanistic descriptions of how a software requirement would create value (the consequences of which were previously discussed in 2.1.2.3). Instead, alternatives can be evaluated against low level causes (e.g., ‘Usability’), and then the value created by those causes can propagate up through the causal chains, e.g., to ‘Sales’.

Following on from Adam Smith, Mill distinguishes value in use, from value in exchange [Ahia03, p.21]. This thesis is primarily interested in value in use, unless the software product is going to be traded or sold (as in Market Driven RE [KDRN07]). Finally, Ilayperuma and Zdravkovic distinguish between two categories of values: Internal (within the minds of people) and Economic [IlZd10]. However, for the purpose of this thesis, the distinction is instead made between Psychological Values and Business Values, given that not all Business Values are Economic.

**Values: Business**

The literature provides numerous lists of ‘good things’ that businesses desire (i.e., business values) which are each somewhat comparable to Frankena’s list of intrinsically good things for individuals (2.1.1.1), in that they are often (perhaps incorrectly) considered to be good without question.

Firstly, Kaplan and Norton’s seminal Balanced Scorecard proposes a set of four ‘perspectives’ (Figure 2.7) for organising business goals, where all of the goals ultimately contribute to the finance perspective [KaNo96a] (as is demonstrated by the causal ordering of the perspectives in Kaplan & Norton’s more recent ‘Strategy Maps’ approach [KaNo04]). This implies that financial goals are the ‘final’ source of value in a chain of instrumental generation for a business (rather than for a person).

![Kaplan & Norton’s Business Goal Perspectives](KaNo96a)

As testament to the completeness of Kaplan & Norton’s four perspectives, Eckartz et al. performed a literature mapping exercise on numerous papers proposing specific benefits of IT [EDWH09], and successfully categorised them according to the Balanced Scorecard’s four perspectives [Ecka12, p.123]. Eckartz et al.’s mapping of benefits also considers abstraction to the three ‘Anthony Triangle’ levels of business decisions (Strategic, Tactical, Operational) [Anth65], as is shown in Table 2.2. However, it should be noted that both in practice and in the literature, most benefits of IT are stated at the operational and tactical levels, since agreement on precisely what constitutes a ‘strategic’ benefit is far less agreed upon [EDWH09, p.5] Also, the proportional majority of implemented IT systems have affected the operational and tactical levels [Thor99, p.26]. Indeed, with reference to Zuboff’s three types of IT system, the benefits of ‘Transformative IT’ are far more complex to calculate than those of ‘Informative IT’, or of ‘Automating IT’ [Zubo85] 10. This is because the first tends to streamline existing manual work activities within few organisational units, while the second changes work processes within many organisation units, and the third changes the nature of the business’ industry [Thor99, p.27].

---

10 Zuboff’s three types of IT system directly map to Anthony’s business decision levels (Transform: Strategic, Inform: Tactical, Automate: Operational) [Thor99, p.14]. Examples include e-Commerce, Customer Information, and Payroll systems, respectively.
Secondly, Reifer’s framework for making a software business case [Reif01] proposes four ‘dimensions of improvement’ (Figure 2.8), where each dimension’s importance is a function of the firm’s current strategic goals. Interestingly, each of Reifer’s dimensions can be encapsulated by Kaplan & Norton’s four business goal perspectives, and so each are also ultimately instrumental for financial goals.

Thirdly and penultimately, Ohno identified types of waste in the product manufacturing business [Ohno88] (Figure 2.9), primarily enumerating ways of avoiding or cutting costs (leading to benefits\(^ {11} \)) at the operational or tactical business levels. Owing to its comprehensiveness, Ohno’s list is now a part of the industry-wide (i.e., not manufacturing-specific) ‘Six Sigma’ management toolset [Isix09].

Finally, Clements and Bass recently performed a clustering exercise after a systematic literature review of ‘business goals’, and consequently proposed ten categories of business goal (Figure 2.10), claiming that “any of an organization’s business goals is likely to fall into at least one of these categories” [ClBa10]. (Although, the categories are defined so abstractly that the claim of

---

inclusiveness seems trivial.) Their aim was to aid the elicitation of business goals in order to better define software architecture, rather than to propose a list of intrinsic values for businesses. As such, their goal categories overlap, and can be categorised under Kaplan & Norton’s four perspectives. Consequently, each of Clements & Bass’s categories can be considered as instrumentally good for ‘Finance’. (While categories #3-6 are people oriented, they are likely to ultimately create financial value e.g., fulfilling responsibilities to employees reduces costs of hiring, knowledge loss, etc.)

<table>
<thead>
<tr>
<th>Clements &amp; Bass’s Generic Business Goal Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Growth and continuity of the organization</td>
</tr>
<tr>
<td>2. Meeting financial objectives</td>
</tr>
<tr>
<td>3. Meeting personal objectives</td>
</tr>
<tr>
<td>4. Meeting responsibility to employees</td>
</tr>
<tr>
<td>5. Meeting responsibility to society</td>
</tr>
<tr>
<td>6. Meeting responsibility to country</td>
</tr>
<tr>
<td>7. Meeting responsibility to shareholders</td>
</tr>
<tr>
<td>8. Managing market position</td>
</tr>
<tr>
<td>9. Improving business processes</td>
</tr>
<tr>
<td>10. Managing quality and reputation of products</td>
</tr>
</tbody>
</table>

Figure 2.10: Generic Business Goal categories, from a literature clustering exercise [ClBa10].

To conclude, the typical business exists primarily to maximise shareholder wealth [BrHo11, p.9]. To be more inclusive of all types of enterprises (e.g., government organisations or charities), Regev and Wegmann propose that the highest level goal of an enterprise is “the wish to survive” [ReWe05]. However, ‘survival’ is less a purpose and more a necessity for something else. Rather, both the Object Management Group (OMG) and Wieringa imply that in any business context, something is valuable only for its ultimate contribution to the achievement of the business’ ‘Mission’ [Wier96, p.95], i.e., that which ‘makes operative’ the ‘Vision’ of the business [Obje10]. Hence, only this ‘Mission’ is not a means to a higher end in a chain of instrumental value generation for the business (and so only it would be considered intrinsically good for the business). However, even achieving the ‘Mission’ of most businesses is instrumentally good for the shareholders’ wealth maximisation, which explains why all of Kaplan & Norton’s business goal perspectives (and hence the majority of the listed business values) contribute toward the ‘Finance’ perspective. Nevertheless, this does not mean to say that benefits of a software requirement should be described in terms of their direct contribution to the ‘Mission’ (or to shareholder wealth maximisation), simply because the software will not directly achieve it. Rather, a mechanistic causal description of a software requirement’s potential for creating value for a business would be in terms of the aforementioned business values, which themselves are widely accepted to be capable of contributing to a business’ ‘Mission’. In summary, the benefits of a software capability should most often not be directly defined financially.

The last point of discussion is that benefits to a business should not be conflated with reasons for a project’s initiation. The IEEE’s PMBOK (Project Management Body Of Knowledge) proposes a list of the latter [Ieee11a, p.75]: Market Demand, Organisational Need, Customer Request, Technological Advance, and Legal Requirement. Such generic project causes are considered by the IEEE as ‘triggers’ for investigating the business case, and not as benefits themselves. For example,

---

12 Other software project initiation causes are proposed in the literature, such as Bocij et al.’s [BoGH08, p.321]. However, such lists tend to be versions of the popular ‘PESTEL’ (Political, Economic, Social, Technological, Environmental, Legal) macro-environmental factors, which are often used as opportunities or threats in business ‘SWOT’ (Strengths, Weaknesses, Opportunities, Threats) analysis [Cipd13]. In this terminology, merely describing the ‘Opportunities’ for a software project is not adequate value analysis.
compliance with a ‘Legal requirement’ is not by itself a benefit, because it may actually increase production costs, reducing the customer base, leading to commercially unviable profit margins.

**Values: Psychological**

There are three popular lists of values from a human psychology viewpoint: Maslow’s hierarchy of needs [Masl54], Schwartz’s personal values [SMLB01], and Holbrook’s consumer values [Holb98]. The development of cultural psychology and marketing (and hence the latter two) has meant that Maslow’s hierarchy has become less popular amongst psychologists [ZdSG13], despite its widespread adoption. Schwartz defines personal values as “desirable, trans-situational goals, varying in importance, that serve as guiding principles in people’s lives.” [SMLB01], and claims that all values previously theorised (including philosophical, religious, financial, etc.), can be classified as belonging to one of the ‘motivationally distinct’ basic values in Figure 2.11.

<table>
<thead>
<tr>
<th>Schwartz’s Personal Values [SMLB01]</th>
<th>(mapped to Holbrook’s Consumer Values [Holb98])</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Achievement — personal success (Excellence)</td>
<td></td>
</tr>
<tr>
<td>2. Benevolence — welfare of people (Spirituality)</td>
<td></td>
</tr>
<tr>
<td>3. Conformity — non-violation of norms/standards (Efficiency)</td>
<td></td>
</tr>
<tr>
<td>4. Hedonism — pleasure (Aesthetics)</td>
<td></td>
</tr>
<tr>
<td>5. Power — status, prestige, control (Status)</td>
<td></td>
</tr>
<tr>
<td>6. Security — safety, stability (Efficiency)</td>
<td></td>
</tr>
<tr>
<td>7. Stimulation — excitement, novelty (Play)</td>
<td></td>
</tr>
<tr>
<td>8. Self-direction — independent thought/actions (Play)</td>
<td></td>
</tr>
<tr>
<td>9. Tradition — respect for customs (Spirituality)</td>
<td></td>
</tr>
<tr>
<td>10. Universalism — protection of people/nature (Ethics)</td>
<td></td>
</tr>
</tbody>
</table>

Similarly, Holbrook defines a consumer value as “an interactive [i.e., consumer ←→ product], relativistic preference experience” that drives human affection of products [Holb98]. Hence, Holbrook’s consumer values can be considered a specialisation of Schwartz’s personal values, which are said to motivate human desires and actions. Recognising this similarity between Holbrook’s consumer values and Schwartz’s personal values, Zdravkovic et al. provide a mapping between the two groups of values, as in Figure 2.11. (Frankena’s list of intrinsically good things, as discussed in Section 2.1.1.1, could also be considered as ‘Internal’ or ‘Psychological’ values, but somewhat overlap).

**Summary Point #23: Implication for the Thesis**

It should be possible to assist the elicitation of the ends (final and non-final) that a software requirement is to be a means to, by using this Section’s (2.1.3.3) lists of values for businesses and individuals, as prompts when interviewing stakeholders.

**2.1.4 Problems and Risks**

In the context of RE, a problem is “a dissatisfaction with the current situation, for which a possibility of improvement exists”, and is owned by a person, group, or institution [Wier96, p.86]. Continuing this line of thought, Jackson and Zave summarise that ‘the business of requirements engineering’ is to describe this dissatisfaction along with a valid solution specification [ZaJa97]. Hence the conceptual framework diagram (Figure 2.1) illustrates that the ‘root’ of all software requirements is the desire to reduce a problematic state of the world. Without problems, there would be no need for software, and so in the context of software engineering, problems in the application domain are desirable.
Risks are essentially problems that do not necessarily currently occur, but might occur in the future. More formally, a risk is “an event with a potentially undesirable outcome whose occurrence has some known, but potentially uncertain probability distribution” [SiNu01]. Contrary to reducing a problem, reducing a risk might never actually make any valuable impact, when evaluated retrospectively. For example, internal software security features (i.e., not including those acting as a deterrent) might reduce the risk of data theft, but if no attempts to steal the data were made in the time period of evaluation, then those features would not have been useful. Indeed, a significant complication in predicting the instrumental value of a risk reduction, is that many risks have very little historic data to inform the construction of probability distributions, especially in safety critical fields such as nuclear power [OBDE06]. Nevertheless, uncertain or low likelihood is often (perhaps incorrectly) considered to be irrelevant in valuations of risk reduction where there could otherwise be severe consequences, as the ‘Pascal’s mugging’ thought experiment illustrates. (The Pascal’s mugging thought experiment is based on the scenario of a mugger offering to swap a victim’s wallet for any large but finite amount of utility: £, time, etc. Regardless of the “very, very” low probability that the mugger would fulfil their promise, the very high utility to be potentially gained makes the offer seem attractive under the popular ‘maximise expected-utility’ decision criteria [Bost09].) Hence, while risk management best-practice considers risk as the product (multiplication) of an undesirable event’s likelihood and its severity [Ieee11a], value modellers must keep in mind that a low likelihood reduction of a severe risk does not necessitate that the software requirement should be considered highly valuable.

Summary Point #24: Implication for the Thesis
Existence of problems, either existing or foreseen (i.e., risks), provide scope for a software requirement to be valuable.

2.1.5 Desires, Goals, & Objectives

Beardsey succinctly remarked that “to connect the desirable with the desired, [is to] connect values with human needs and wishes” [Bear65, p.17], where the former have already been described in Section 2.1.3.3, and the latter are—in this context—ultimately derived to requirements on software (described later in Section 2.1.5.3). Premature connection between needs and wishes with intrinsic values (i.e., not exploring the extrinsic value of desires) encourages the belief that desires, which “provide the data and conditions of value-problems, and set some of the limits within which solutions are to be found”, are more final than they really are [Bear65, p.17]. In other words, the goals and objectives that define the comparison criteria and bound the search space for the software solution (i.e., the requirements), should not always be considered immutable, and may otherwise lead to valueless requirements.

However, in a typical organisation, the desire to ‘implement software X’ would not be easily changed by less senior employees (e.g., RE practitioners). Indeed, the risk of appearing insubordinate may be off-putting to the practitioner, especially if they are to be evaluated purely by ‘is software X implemented?’ Therefore, organisations with a culture less afraid of challenging decisions for the purpose of value maximisation, would likely gain competitive advantage (as in ‘emergent strategy’ in Section 2.1.5.3).

Summary Point #25: Implication for the Thesis
Software project goals (for which software requirements might be instrumentally valuable for) will rarely be intrinsically good. However, in the context of an organisation where responsibility is bounded, the project goals might be sufficient exploration of extrinsic value, in the sense that if satisfied, job security is not threatened—at the expense of value creation.
2.1.5.1 Desires

Desires are the fundamental motivation for goals, needs, requirements, and indeed for all human actions [LiFo11, p.347]. More precisely, to ‘desire’ is “to express a wish to obtain” [Dict14], implying that the object of desire is not currently obtained, and hence that there exists a less desirable (i.e., problematic) state of the world to be solved. To contradict this, the reader might argue that it is possible to desire something already obtained, hence challenging this thesis’ conceptual model’s assertion that problems are necessary for the existence of desired ends (Figure 2.1). However, one must distinguish affection for an obtained thing (e.g., a software user likes their software’s current ‘print’ function), and desire for an un-obtained or un-guaranteed thing (e.g., a software user would like a ‘print’ function in future versions of the software). In support of this distinction is the KAOS meta-model’s definition of a ‘Wish’ (synonymous with ‘Desire’ [Dict14]) as the meta-relationship: “Wish(ag, G) if human agent ag wants goal G to be achieved” [DaLF93], which implies that wished for goals are not assumed to be achieved in the current or future state of the world. Nevertheless, it is not uncommon for stakeholders to propose software requirements that enable capabilities already existing in the application domain, due to their fear of losing those capabilities. This causes ambiguity about the value of the software project’s requirements: i.e., which ones would cause the expected benefits?

Summary Point #26: Implication for the Thesis
Exploration of the instrumental value of a requirement should concern change in the application domain, i.e., achievement of desired states of the world not currently existing, states of the world or not guaranteed to exist in the future.

Summary Point #27: Implication for the Thesis
In practice, politics or organisational hierarchy make it seem as though some stakeholder requirements ought to be satisfied ‘for their own sole sake’. However, this obstructs a key aim of value analysis: exploration of more valuable alternatives.

Desires: Compared to Needs

Desires differ from needs, in that a manager may desire an information system, even if they really need a change in organisational processes [Wier96, p.22]. Gause and Weinberg claim that a client’s needs can never truly be known, and so requirements engineering is, practically speaking, about...
determining desires [GaWe89]. Conversely, Wieringa proposes that since there should be agreement between the software developer’s perception of the client’s needs and the client’s actual desires, then “user needs are user desires” [Wier96, p.67], hence defining a need as a “lack of something desirable” [Wier96, p.22]. However, compared to Gause and Weinberg’s definition, Wieringa’s is ideological and does not differentiate a client’s known desires from their unknown (but perhaps more optimal) desires. Along these lines, the SEBOK cleanly distinguishes between ‘perceived needs’ (i.e., desires) and ‘real needs’ (i.e., needs) [PyOl13, p.222]. Finally, it is worth noting that desires change over time, as the marketplace of solutions advances, whereas needs are often more timeless. E.g., the changed desire of “Microsoft Office 365” from “Lotus Notes” are both linked to the need for “Email communication”.

Summary Point #28: Implication for the Thesis

Given that desires are more fickle than needs, models of a requirement’s value should aim to explain the ‘real need’ behind the ‘perceived need’ that the requirement may represent. Only then can their pertinence be confidently assured, and the full potential of retrieving & re-using value models be realised (due to increased ‘recall’ of similar desires, w.r.t. “precision & recall”).

Desires: Current versus Future

Neither software development nor procurement is instant. Stakeholder desires that were elicited at the start of a project are likely to evolve by the time the resulting software requirements are implemented. This gap will be larger with software projects in fast-changing environments, or with those following ‘Big Requirements Up Front’, (i.e., waterfall) development [Ambl12]. Hence, future (uncertain) desires and their value must be predicted, rather than only describing existing desires.

While there can be more certainty in expected benefits referencing the current state of the world, due to the ability to investigate rather than speculate problems, this certainty comes at the expense of confidence. For example, ‘current’ desires can be radically re-shaped by disruptive technologies (as explained by Christensen’s ‘Innovator’s Dilemma’ [Chri13]). In other words, developing products to fulfil current desires is like “skating where the puck is now, rather than where it will be” [Chri13].

Perhaps the most famous example is the quote attributed (albeit wrongly [Vlas11]) to Henry Ford: “If I had asked people what they wanted, they would have said faster horses”. (While this quote is often used in defence of ignoring stakeholder input, a wiser interpretation would be that requirements engineers should enquire about the reasons why alternatives to current desires — ‘horses’ — would be unsuitable, e.g., ‘a motor car would be unsuitable for dressage’ leads to a new requirement.)

Summary Point #29: Implication for the Thesis

Value models should concern the future desires of stakeholders, i.e., the period of time when the software-to-be will exist.

One might use examples of disruptive technology to argue that technology driven software requirements (i.e., those without existing demand) are desirable, and that the problems to be solved will be found after the implementation. E.g., as with the electronics that lead to digital watches [NoVe14], in turn creating awareness of the problem that ‘mechanical watches can be inaccurate’. In other words, one might argue that the instrumental value of a software requirement is not an important concern in the RE phase, since unanticipated value may be later derived from the software capabilities. However, it is irrational (due to weak analogy — Section 2.1.2.4) to argue that an ‘internet connected toaster’ will be successful in the same way that a motor car was successful, just because they have in common that at the time of their conception, the need for them was not felt by consumers. Successful software requirements, disruptive or not, will, by definition, alleviate a problem. If the mechanism by which that would occur cannot be explained, then the risk that it would provide poor value is significant — despite that it is not impossible.
Summary Point #30: Implication for the Thesis
A software requirement whose instrumental value cannot be explained in terms of acknowledged problems, will not necessarily have zero value (benefits may be recognised after the capability has been developed), but it does indicate risk.

Finally, given that the implementation of a software requirement can be considered as an investment [SiNu01], it is important to consider the existence of the empirically verified [GhSV05] risk-reward trade-off [Inve14a]. Consequently, the most valuable software requirements may be those whose sponsoring stakeholders have the highest value expectations of, but also have relatively low confidence in their realisation, e.g., if they provide un-tested novelty to the software product (for competitive advantage), as opposed to more common but well-tested capabilities [KSTT84].

Summary Point #31: Implication for the Thesis
Due to the risk-reward relationship, low stakeholder confidence in a requirement’s value should not always be avoided.

2.1.5.2 Goals

A goal “is simply a desired situation” [Wier96], “something that one wishes to achieve” [KaLo03], or “something a stakeholder wants to be true of the world” [Emma13]. Indeed, according to Engelsman and Wieringa’s review of GORE (Goal Oriented Requirements Engineering) languages, a ‘goal’ tends to be defined as a desire or an intention of a stakeholder [EnWi12] — i.e., as a higher level stakeholder oriented construct distinct from software requirements. A key perspective within GORE, however, is that system requirements are essentially considered to be stakeholder goals restated at a low level of abstraction [BlCV04], e.g., in KAOS, a requirement is a goal assigned to one software agent [Lams09a]. In the words of Kavakli’s unifying GORE framework, ‘requirements implement goals in the same way that programs implement design specifications’ [Kava02], and so in GORE, requirements are refined from goals and are evaluated by their ability to satisfy them. Hence, goals can either prescribe properties of the system to be developed, or properties of the environment (this distinction is indicated by qualifiers such as ‘system’ or ‘business’ goals). As such, links connecting requirements to goals could form a network to explain the instrumental value of a requirement (as is more thoroughly evaluated in the review of approaches located later in this thesis, in Chapter 4).

Goals are distinct from desires primarily in the implied resource commitment entailed by a goal. In other words, “If you want to separate a company’s goals from its desires, look at what they are committing resources to achieve” [Wier13]. (Or, as Exupéry put it, “a goal without a plan is just a wish” [Sain39].) An additional difference is that responsibility for satisfying goals must be assigned to actors who are capable of fulfilling the goal [Lams09a], while desires can remain unassigned or un-assignable. As a simple illustration, the problem that ‘the sun is not shining’ cannot be inverted to form a goal ‘sun be shining’, since responsibility for this goal is unassignable. In order to achieve that desire, it must be abstracted to a higher level treatable outcome of that untreatable problem, e.g., ‘it is dark’, and then inverted, e.g., to goal ‘not be dark’. Finally, extensive behavioural economic and psychology literature shows that “setting a goal has some key properties: an increase in effort (energy put into the task), persistence (time spent on the task), attention (more focus on the task with the set goal and away from non-relevant activities) and it encourages the use and acquisition of knowledge” [Smit15], and so prescriptively modelling the instrumental value of requirements by setting causally interlinked goals, is likely to have positive motivational and value realisation effects.

Summary Point #32: Implication for the Thesis
Goals are desires with at least some resource commitment, and so are a preferred (and sometimes pre-existing) concept for reasoning about a requirements’ value, given that desires that don’t get funded will waste RE effort.
Numerous authoritative definitions for ‘goal’ are unfortunately conflated with ‘objective’, e.g.:

- ISO 29148: “The term ‘Goal’ (sometimes called ‘business concern’ or ‘critical success factor’) refers to the overall, high-level objectives of the system. Goals provide the motivation for a system but are often vaguely formulated” [Ieee11b];
- van Lamsweerde’s: “A goal is an objective the system under consideration should achieve. Goal formulations thus refer to intended properties to be ensured; they are optative statements as opposed to indicative ones, and bounded by the subject matter” [Lams01].

However, the OMG’s Business Motivation Model (BMM) defines an objective as a specific “statement of an attainable, time-targeted, and measurable target that the enterprise seeks to meet in order to achieve its goals” [Obje10]. According to the definitions of goals and objectives provided by the BMM, the difference between a goal and an objective lies in the goal’s hardness: That is, whether or not the goal’s (partial or full) satisfaction can be determined in a clear cut (unambiguous) manner. (Hence ‘hard goal’ and ‘objective’ are interchangeable, and are opposite to ‘soft goal’). Lastly, it is widely recommended to “strive to convert soft goals to hard goals by agreeing criteria for agreeing their fulfilment; otherwise how could a project conclude?” [Bone11, p.34].

Summary Point #33: Implication for the Thesis
Prescriptive statements of desire are less ambiguous when they are specified as objectives, as the OMG’s BMM metamodel defines them (i.e., numerical and testable targets). This reduction in ambiguity extends to estimates of a requirements’ value.

2.1.5.3 Business Objectives & Business Strategy

In the context of this thesis, a ‘business objective’ is an objective that has resource commitment (or sponsorship, endorsement, etc.) from important stakeholder(s) — typically from the business that will own the software. *(The term ‘business objective’ is not restricted to software for use in a business, in the same way that the term ‘business logic’ of a software program [Chus12, p.45] is used to describe the application domain logic. Furthermore, they might prescribe a certain extent of realisation of the values described in Section 2.1.3.3, not limited to the ‘Business’ values).*

Given the widespread popularity of the ‘business objectives’ term (a Google search in 2015 finds 7.4 million results, compared to 0.5m for ‘expected benefits’), and as shown in the conceptual framework diagram (Figure 2.1), a ‘business objective’ is the construct adopted by this thesis’ conceptual framework for modelling individual expected benefits (ends) of software requirements. Extra advantage is gained due to the popularisation of the ‘SMART’ quality criteria for business objectives [Dora81], because many practitioners with basic business education will already know that objectives should be ‘hard goals’ due to SMART’s ‘Specific’ and ‘Measurable’ criteria. (Hence exploiting the practitioners’ existing knowledge to improve usability — an ‘affordance’ [Norm13, p.11]). Finally, ‘expected benefits’ can be considered synonymous with ‘business objectives’, but only when an objective is referred to in the context of being an end in a means-end pair (in this context, the means being either a software requirement or another objective).

Summary Point #34: Implication for the Thesis
When discussing, formalising, and modelling the ends that a software requirement is a means to, the requirements engineer may find success in eliciting ‘hard’ goals if the ends are referred to as ‘business objectives,’ due to the popular SMART criteria.

Given that exploring the instrumental value of software requirements in a business context will almost certainly entail traceability to business objectives, the following paragraph argues that
traceability to business strategy\textsuperscript{13} can also be claimed. An authority on codifying business strategy, the OMG’s BMM states that 1) an objective quantifies a goal, 2) a goal is a desired result, 3) courses of action are designed to achieve desired results, and 4) strategies are courses of action [ObjEC10]. Hence, an objective may have one or many possible strategies to achieve it, in the same way that one end can have many possible means to achieve it. However, due to the possibility of multiple levels of goal/objective abstraction, objectives and strategies relationships may be bidirectional, i.e., Parent(objective)→Child(strategy) as well as Parent(strategy)→Child(objective), since a strategy may be comprised of objectives. Indeed, inspired by Porters’ ‘activity-system mappings’ modelling notation, Bleistein et al. propose that “business strategy can be represented in terms of a collection of high-level abstract business objectives” [BACR03]. In other words, “Business goals are further refined by strategies. The end result of applying strategies is more concrete goals” [MBHO10].

**Summary Point #35: Implication for the Thesis**

If the instrumental value of a software requirement is modelled such that it is traced to an organisation’s various levels of business objectives, then it can be claimed that the requirement is traced to (and hence is a component of) business strategy.

### Business Strategic Planning: Similarity to Requirements Engineering

It is pertinent to briefly mention the process by which strategies (means) are formulated for business objectives (ends), given the conceptual similarity to the process of deriving software requirements from desires, and hence constructing the necessary substrate for modelling their instrumental value. ‘Strategic planning’ is concerned with the breaking down of a goal into formal steps, so that they can be implemented “almost automatically” [Mint00]. Strategic thinkers typically devise (synthesize) goals, which are then decomposed (analysed) by strategic planners. Strategy analysis (sometimes referred to as strategic programming) includes codification, elaboration, and conversion [Mint00], e.g., converting a supermarket’s goal to ‘expand into shopping centres’ to the when, what, where, how, and how many. Porter advocates the use of analytical (top-down) techniques for strategy development [Port87], but Mintzberg warns that an organisation should only follow an analysed strategy when it is sure of its environment’s stability, otherwise strategies should remain flexible to adapt to a changing environment, i.e., ‘intended’ vs. ‘emergent’ strategies [MiWa85]. (A major difference between RE and strategic planning, is that while much has been written on the theory of business strategy, ways to represent it (e.g., modelling) have received little attention [WaGh06].)

**Summary Point #36: Implication for the Thesis**

A chain of means-ends representing a software requirement’s instrumental value, is one possible strategy for achieving that ultimate value. Its validity will decrease as the time between modelling and implementing increases, as the environment changes.

#### 2.1.6 Requirements (User, Stakeholder, System, Software, etc.)

There is little consensus on the precise meaning of the terminology used in RE — especially on the term ‘requirement’. Jureta et al. recently summarised that it is “variously understood as (describing) a purpose, a need, a goal, a functionality, a constraint, a quality, a behavior, a service, a condition, or a capability” [JuMF08]. Fifteen years before that, Harwell et al. asserted that “if it mandates that something must be accomplished, transformed, produced, or provided, it is a requirement — period” [HAHM93]. What both of these definitions share in common is that a requirement is either about a problem (i.e., a purpose, need, or goal to be accomplished) or about a solution (i.e., a functionality, Business strategy is “both the rationale for and the means by which a business organization competes with industry rivals” [Port96], inherently necessitating the existence of an end for which it is a means to.
quality, behavior, service, or capability to be transformed, produced, or provided). As Wieringa puts it, there are “two schools of thought” in RE [Wier05], and confusion among practitioners about which ‘school’ their requirements should adhere to [Alex02a] seems inevitable due to the semantic overload of the ‘requirement’ term. Idealistic and ambiguous guidelines, such as that requirements should describe what software must do rather than how it will do it, only complicate the issue [Davi93, p.17], since as Kovitz eloquently puts it, “everything that a piece of software does is what it does, and everything that a piece of software does is how it does something” [Kovi98, p.51]. Qualifying adjectives, such as those in this section’s heading, are often prepended to ‘requirement’ to indicate which ‘school of thought’ the requirement belongs. Popular requirements [HuJD11], systems [Bued00, SBJA98], and software [Somm11a] engineering textbooks, along with international systems and software requirements engineering standards [Ieee11b, Iiba09, PyOl13] primarily refer to two such qualifiers: stakeholder and system. (A more comprehensive literature mapping and discussion of requirements types can be found in one of this researchers’ papers in the appendix [ELDK14].)

2.1.6.1 Requirements: Stakeholder (& User)

Since a stakeholder is “an individual, group of people, organisation, or other entity that has a direct or indirect interest in a system” [HuJD11], the popular business, legal, and user requirement qualifiers could be considered as specific types of stakeholder requirement. Confusingly, some refer to stakeholder requirements as business requirements, but the latter specifically describes the needs of the enterprise rather than those of any other particular (or class of) stakeholder [Ieee11b, p.42]. Another distinction is of requirements placed upon stakeholders to be causally responsible for something (‘expectations’ [Lams09a, p.490]), rather than requirements wished for by stakeholders ([Lams09a, p.402]).

Summary Point #37: Implication for the Thesis

‘Business’ or ‘stakeholder’ requirements may be synonymous with goals and objectives, depending on the terminology used by the RE practitioners. Hence, it is crucial to clarify which type of ‘requirements’ are being assessed for value.

2.1.6.2 Requirements: Software (& System)

Since a system is “a collection of components – machine, software, and human, which co-operate in an organised way to achieve some desired result” [HuJD11], there can be requirements on the whole system, its components (in a software intensive system, primarily software), and on its interfaces. Hence, a software requirement is a system requirement by definition, but since “a system does not merely consist of software” [Lams09a, p.19], the converse is not true. Whether or not the system requirements contain non-software requirements (e.g., on expected human, hardware, or interface behaviour) depends on whether the software will provide “essentially all the functionality”, or whether it will be just one “part of a larger system” [Ieee11b, p.45].

Summary Point #38: Implication for the Thesis

‘Software’ requirements are a type of system requirement in the solution domain (i.e., what is to be constructed, rather than what is given), which specify world phenomena that the machine should help satisfy, not machine phenomena [Jack97, p.7].

However, the precise definition of a ‘software requirement’ is ambiguous in the literature. The IEEE’s software engineering glossary defines only the unqualified term ‘requirement’ as “a condition or capability that must be met or possessed by a system or system component” [Ieee90, p.62]. According to this definition, a ‘stakeholder requirement’ may also be considered a ‘software requirement’ if it is a condition ultimately ‘met’ by a system component (i.e., the software). van Lamsweerde provides a clarifying distinction, stating that a software requirement is to be (causally) enforced “solely by
the software to-be” [Lams09a, p.19]. This is a pragmatic and logical restriction on the term, since most ‘stakeholder requirements’ would likely require the cooperation of many agents for their satisfaction, e.g., changes in business processes. (One might also note that software requirements cannot be enforced without hardware, electricity, and cooperation between an almost endless list of agents necessary for those, and so, the ‘software’ agent is assumed to be a composition of these parts.)

While van Lamsweerde’s definition of ‘software requirements’ may be amongst the most descriptive and hence useful, it is likely to be interpreted more loosely in practice, e.g., as ‘software system requirements’. Indeed, ISO 29148’s definition of a ‘software requirements specification’ (a “structured collection of the requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces” [Ieee11b, p.6]), does not include van Lamsweerde’s software-agent restriction, and so specifying a function of the software seems to make it a ‘software requirement’ in ISO 29148 parlance, regardless of whether the function’s operation might require cooperation with non-software agents. In summary, for the purposes of modelling instrumental value, it is permissible to model a ‘software requirement’ according to ISO 29148, but if it does not comply with van Lamsweerde’s definition, it indicates that the requirement has not been sufficiently decomposed.

**Summary Point #39: Implication for the Thesis**
When modelling the value of a ‘software requirement’, the requirement should be causally enforced solely by a new or existing software agent (component). Otherwise, the software requirement has not been adequately decomposed, indicating incompleteness, and therefore that unknown causes (means) for the desired effect (end) can and should be prescribed.

### 2.1.6.3 Requirements: Functionalities (Behaviours) & Non-functionalities (Qualities)

Whereas problem-oriented requirements consist of “problematic phenomena and goals to be achieved”, solution-oriented requirements consist of “functions and quality attributes of a desired solution” [Wier05, p.305], i.e., ‘Functional Requirements’ (FRs) and ‘Non-Functional Requirements’ (NFRs) [RoRo12a], respectively. FRs are behavioural requirements “that specify the inputs (stimuli) to the system, the outputs (responses) from the system, and behavioural relationships between them”, [Glin07] and NFRs are “nonbehavioral aspects of a system, capturing the properties and constraints under which a system must operate” [Anto97].

Regardless of the myriad of available (and often contradictory) definitions for FRs and NFRs, Glinz demonstrates that the distinction lies in the requirement’s representation rather than its concern [Glin07]. For example, the different requirement representations ‘…require username and password’ (FR) and ‘…probability of unauthorised access’ (NFR) refer to the same security concern, but provide different information about it. The difference in representation has important consequences for the analysis of the requirement’s value, both in its outcome, and in its opportunity costs (given that the alternatives may be different). For example, alternatives to the former (FR) might include ‘require RSA key’, ‘require verified email address’, ‘require social media account’, etc., while the latter (NFR) might vary in the probability of unauthorised access, e.g., ‘{1, 5, 10, 20} fraudulent logins per year’. The modelled instrumental value (outcomes) of these requirement representations would also vary, e.g., from ‘increasing perceived security’, ‘increasing authorisation process usability’, etc., to ‘reducing the rate of user-base loss due to perceived insecurity by {0.1, 0.2, 1, 2}% per year’. (More detailed technical implications of these requirements types are discussed in-line with their modelling guidance later in the later description of this thesis’ framework in Chapter 5.)

**Summary Point #40: Implication for the Thesis**
The representation of a requirement (e.g., its specification as functional or non-functional) affects the way in which its value is interpreted. Hence eliciting expected benefits from stakeholders, rather than purely inferring them from requirements, is crucial.
Finally, Wieringa makes the important distinction that non-behavioral specifications of system properties (NFRs) often depend on groups of people agreeing on the presence or absence of these properties, “without using experimental evidence that unambiguously settles the question whether a system has the property” [Wier05, p.21]. Different groups may agree and disagree with the user-friendliness of some software product interface, just as they might about the quality of a painting, making their presence subjectively determined. Hence, it is often useful to define proxy behavioural specifications for such properties, “whose presence indicates the presence of the nonbehaviorally specified property, but which are not equal to it” [Wier05, p.21]. e.g., the previously mentioned ‘probability of unauthorised access’ NFR representation of the ‘security’ concern. Hence, it is crucial that such quantified NFRs (considered best practice [Maid06]) acting as proxies, also describe the actual intended property of the system, in order to avoid and resolve mismatches between requirements concerns and representations. On a similar theme, Kaindl draws a parallel between RE and Plato & Socrates’ allegory of the cave, given that “the same way that believing that a shadow of a brick is a brick itself might be a problem when building a house; believing that a requirement representation is a requirement itself might be a problem when building a system” [KaSv10]. A simple practical implication is that a ‘requirement itself’ may be volatile, while the representation of that requirement is a snapshot of that requirement at that particular time and context, and will not change unless that change is elicited and made explicit.

Summary Point #41: Implication for the Thesis

‘Actual’ software requirements are intangible, and so their specifications are merely representations or proxies of them. It is important not to confuse a desired requirement with a proxy designed to indicate its presence, since misalignment can occur.

2.1.6.4 Requirements: Modelling Perspectives

Finally, a useful distinction is Pohl’s four worlds in software requirements engineering (Figure 2.12), each associated with groups of stakeholders (and hence expertise, languages, interests, etc.) who should be included in RE decisions [Pohl96, p.16].

![Figure 2.12: Pohl's Four Worlds in System Modelling for Requirements Engineering](Pohl96, p.16).

Crucially, a requirement has different interpretations in these different worlds, especially between the usage world (where requirements imply provision of a hopefully valuable service to a user) and the development world (where requirements imply implementation tasks [DeNP12]).

Summary Point #42: Implication for the Thesis

Modelling the instrumental value of a requirement mandates that the requirement is viewed in the ‘usage world’, since it is there that it will generate value. This has implications for modelling and metadata, e.g., the agent that ‘owns’ the requirement.
2.1.7 Strategic Alignment

Information Technology is strategically aligned “when a business organization’s goals and activities are in harmony with the information systems that support them” [McSm03]. ‘Alignment’ is often referred to as fit, integration, bridge, harmony, fusion, and linkage [AJPW04], while ‘Strategic’ refers to the assessment of future, rather than current needs (such as resource allocation or effectiveness and efficiency) — as in ‘tactical’ or ‘operational’ alignment [UmST11]. While there is no widespread consensus as to what precisely constitutes alignment [KeKa05], it is “almost always viewed as a binary relation relating a [parent] strategy to another [child strategy]” [WaGh06]. These strategies are usually ‘Business’ and ‘IT’ (or otherwise HR, R&D, etc.), and are usually determined to be “in or out of alignment according to a snapshot in time of an organization’s strategic intent” [AJPW04].

Summary Point #43: Implication for the Thesis
A software based system’s strategic alignment is typically considered as ‘binary’. By postulating probabilistic causal propositions (Section 2.1.2.1) about the contribution of a software capability to a desired end, the ‘extent’ of alignment could be ascertained.

Overall, the definition of strategic alignment considered most relevant for this thesis’ conceptual framework is Singh and Woo’s, which describes it as “the synergy between strategic business goals and IT goals” [SiWo09]. Some literature expands the concept of alignment to concern intentional elements other than goals, such as Visions, Missions, Objectives, Strategies, and Tactics (VMOST [Sond99]), as in the OMG’s Business Motivation Model [Obje10]. However, these are merely types of goal (according to Section 2.1.5.2’s definition) defined at varying levels of time and solution-space abstraction, and are likely to complicate communicating value models to non-business stakeholders. Communicability is a primary concern in RE, and the time/solution-space concerns can be dealt with without having to include the extra VMOST concepts in value models (e.g., by Summary Point #46).

Summary Point #44: Implication for the Thesis
Strategic alignment can be modelled through goals at different abstraction levels (as defined and related to value in Section 2.1.5.2). While the intention behind including concepts like Visions (VMOST [Sond99]) is valid, it is likely to obfuscate RE models.

Fonvielle and Carr’s visualisation of top-down alignment (Figure 2.13) is a simplistic but useful introduction to the concept of alignment (albeit generic and individual-oriented, rather than IT-oriented). Crucially, it recognises multiple levels of organisational strategy (and hence the capacity for each level to have ‘strategic’, ‘tactical’, and ‘operational’ [Anth65] goals), as opposed to the more classical interpretation of ‘strategic goals’ as strictly those that affect “the organisation as a whole” [Daft03]. Alignment is a deceptively non-trivial task, especially given that each of the levels shown in Figure 2.13 have typically been designed to encourage group skill specialisation (i.e., high cohesion w.r.t. coupling and cohesion [Ieee90, p.17]), where there are strong differences in domains, language, and interests between alignment levels. This complicates attaining alignment and hence attaining ‘Line of Sight’ (when individuals can relate their actions to the organisation’s strategic goals), and therefore realisation of proven benefits that come with it, such as increased job satisfaction, timely decisions, commitment, and so on [BoBo01].


2.1 Conceptual Framework (Concepts and their Relationships)
Conceptual Framework (Concepts and their Relationships)

Summary Point #45: Implication for the Thesis

The degree to which the value generated by a software requirement supports the value desired by an organisation, can be referred to as its ‘strategic alignment’. Ensuring it is nontrivial in organisations, especially due to role fragmentation. However, it has been established that ‘line of sight’ from day-to-day activities to long term goals is beneficial, e.g., for motivating teams.

Electronic Engineering

Summary Point #46: Implication for the Thesis

Especially in an organisation context, value realisation timescales at the top of a means-ends chain (with software requirements at the bottom) should be longer term than those toward the bottom (shorter term), reflecting the concept of strategic planning.

Despite that strategic alignment is proven to be important for realising value from IT, and to this date remains a non-trivial problem [DeLu14], issues of and techniques for business-IT alignment in software requirements engineering is “almost entirely ignored in RE research literature” [BICV06], (as later shown to still be the case in the review of the state of the art approaches in Chapter 4).

It appears as though the mainstream RE community is satisfied with placing an upper limit on their focus to ‘software project goals’, and indeed, in most cases, alignment inadequacies would likely be blamed on business managers or strategists, rather than on software development teams.
Even within the field of strategic alignment research, there are numerous approaches proposed for validating IT-business alignment (e.g., [HeVe91, LuPB99]), but these tend to focus on management issues and ignore connections to system requirements.

At a more operational software engineering level, Wieringa et al. propose that the problem of Business-IT alignment comprises more than problems of aligning “business processes, needs, added value, money, norms, values etc.”, which exist only in the social world. In total, three worlds must be aligned: the linguistic, social, and physical worlds [WBFG03]. For example, software (in the linguistic world) must be aligned to people (in the social world) by ensuring that meaning attached by people to symbols at user interfaces agrees with the software’s manipulations of them.

Finally, it should be noted that success in attaining strategic alignment is desirable, but it does not guarantee success. E.g., if successful realisation of the strategic objectives fails to lead to the desired end(s). (This is related to the discussion on ‘Unvalidated Theories’ of causality in Section 2.1.2.4.)

2.1.8 Requirements Traceability Links

Gotel and Finkelstein provide the most popular definition of requirements traceability [GoFi94] “Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases”. When a requirement is modelled as a means to an end, explicit traceability is established from a requirement to information that explains ‘its origins’ in the form of the rationale for its existence. Hence, best practices from traceability [CIGZ12], traceability visualisation [LiMa12, MeJD11], and rationale management [DMP06] are relevant to this thesis.

Hull et al. [HuJD11, p.15] summarise the analyses enabled by explicit traceability (and hence those that would be aided by treating this thesis’ problem) as either:

- **Impact analysis** (for changes), e.g., ‘What if this requirement is changed?’
- **Derivation analysis** (for cost/benefit), e.g., ‘Why is this requirement here?’
- **Coverage analysis** (for progress & accountability), e.g., ‘Are all customer concerns covered?’

While numerous taxonomies of requirements traceability relationship types have been proposed [Rsa09], the majority are oriented around hierarchical decomposition and conflict management (as is discussed in more detail in Chapter 4’s evaluation of existing approaches to this thesis’ problem). Those concerned with aspects of causality tend to adopt the primitive ‘necessary cause’ definition of causality, e.g.: “‘Requires’ relationship: ‘requirement R1 can be seen as a precondition for R2’” [Rsa09, p.33], which is considered to be insufficient for value modelling, as discussed in 2.1.2.1.

**Summary Point #47: Implication for the Thesis**

Modelling the instrumental value of a software requirement, is, in the relational aspects, conceptually similar to traceability, and hence best practice techniques for its construction, management, visualisation, etc. are applicable to this thesis’ problem.

2.1.9 Precision, Significance, Accuracy, & Uncertainty

An estimate about the causal strength of a means (i.e., the value contribution from a software requirement to an objective) must balance **significance** with **precision** (as illustrated by Figure 2.15), as well as precision with **accuracy**. Increases in a prediction’s precision tend to reduce the likelihood of the prediction being correct, (i.e., its accuracy), while reductions in a prediction’s precision reduce the usefulness of the prediction by increasing its uncertainty. For example, a precise value estimate that predicts a software feature to provide a ‘326 minute saving on average per process instantiation’
is likely to be inaccurate, while ‘between 1 minute and 600 minutes’ is likely to be accurate, but is of insufficient significance to most practical uses of the prediction. Balancing these characteristics is not trivial, given the costs of precision errors in time and effort, or of poor decision making.

Figure 2.15: Precision versus Significance in real world decisions (image adapted from [Matl14]), or in Cameron’s words: “not everything that can be counted counts...” [Came63].

A memorable example of precision vs. significance is provided by King in a discussion on ISO 8000 (data quality): the precision of an estimate for the distance to the nearest post office depends on whether it is to be used for laying a cable, or providing directions to a traveller [King13].

2.1.9.1 Significance versus Precision

Earlier discussion on probabilistic causal propositions (Section 2.1.2.1) highlighted the need for understanding the strength of a causal component for an effect (Summary Point #10), for which the scientific community has largely chosen probability as the description language [OhOa04, PaWe01]. (As opposed to the common alternative of textual labels such as ‘strong’ or ‘very likely’, which are widely documented to be interpersonally ambiguous [OBDE06, p.85].) However, given the informal and non-quantitative current state of practice in reasoning about the value contribution of software requirements (as argued in Chapters 4 & 6), such quantitative modelling and reasoning may be considered arduous or excessive by software development practitioners. Indeed, probability is known to be widely misunderstood amongst scientists and practitioners [Gige11, Karl02], and furthermore, a recent survey of over 26,000 software developers found that while 48% did not have an IT related degree, less than 4% had a statistics or mathematics background [Stac15].

Perhaps the strongest argument against quantitative reasoning is that conclusions can often be reached with qualitative reasoning; As Chaudhri puts it, one does not need differential equations to conclude that ‘heat + kettle + water → boiling’ [Chau11]. This is in contrast to the many glib quotes often used by proponents of quantification in RE, such as Lord Kelvin’s claim that “when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind” [Thom89, p.73] (as quoted by Gilb [Gilb05a, p.33]). In Chaudhri’s case, the precision afforded by quantitative reasoning exceeds that required for the significance of the knowledge problem. However, the RE context has numerous distinguishing factors that weaken the applicability of this analogy, including:

1. Most decisions that people make are for themselves [GrKü09], whereby:
   i. value judgements are often trivial, highly personal, and required instantly [Wick12];
   ii. other people’s values are not especially important (so risks of conflicts are low);
   iii. instinct or intuition often suffices (low risk in making an incorrect decision).

Conversely, requirements specification is most often performed for systems owned by people other than the requirements engineer, and often has the opposite characteristics to the above.
Figure 2.16: Day to day life decisions are not considered significant enough to warrant high precision, or even conscious decision analysis. However, suboptimal decisions in Requirements Engineering can cost a lot more. (Figure from [KaST82, p.479]; © 1979 ‘The New Yorker’).

2. One can be confident of the universal and timeless assumptions that entail the wide applicability of the qualitative causal proposition that \( \text{heat} + \text{water} \rightarrow \text{boiling} \). To assume a similar level of applicability for past-proven causal propositions in different sociotechnical contexts is to fall victim to many of the common fallacies and misconceptions discussed earlier in Section 2.1.2.4. (It cannot be assumed that people will desire, behave, and interact with software features to the same extents as in the past.) Instead, representing causal propositions about quantitative improvements in terms of concrete units (e.g., ‘minutes’ not %’s) entails understanding the current situation (i.e., benchmarking [Hubb10]), in the particular application domain context.

3. Software functions are defined "in the derivative sense of useful behaviour" [Wier96]. Hence if a functional requirement has been prescribed, then the question is not whether someone believes that it would be useful (obsolete requirements [WnGZ13] aside), but rather whether it will be to the desired extent. Representing the requirement’s value proposition qualitatively reduces the potential for others to challenge its correctness or incorrect assumptions underpinning it. The precision more inherent in quantitative reasoning encourages analytical thought [WiWe96], specificity rather than abstraction [Chau11, p.8], refutability [Popp63], and overall more mechanistic descriptions of causal propositions (Section 2.1.2.3). Indeed, Hubbard concludes that when setting goals, the biggest benefit of eliciting the goal’s associated measures is to find out precisely what is to be addressed, regardless of whether any actual measurement will ever occur [Hubb10]. (On this theme, numerous authors claim that the benefits of quantification is often in ‘the journey rather than the destination’, i.e., the refinement of vague to precise, and validation with stakeholders [Galw04, p.39, Mclu03, p.4, Somm07b, ThHu04].)

The level of precision suitable for a requirement→value causal proposition ultimately depends on the gap in estimated net benefit between the alternatives (including the ‘do not implement’ option). Increased precision for its own sake is wasteful, given that “One cannot obtain a "correct" model by excessive elaboration. On the contrary following William of Occam he should seek an economical description of natural phenomena.” [Box76].
Summary Point #48: Implication for the Thesis

‘Requirements→value’ causal propositions would frequently benefit from the precision afforded by quantitative reasoning, but its simplicity (and hence usability) should be optimised, given that the typical practitioner’s expertise is software, not mathematics.

Finally, it is important to be aware that precision in just the cause, or just in the effect, may not be useful. Take, for example, the advertisement that “our air filters increase engine air flow by 50%, improving engine performance”. While the numerical precision in the cause (+50% air flow) might improve sales of the air filters, the ambiguity in the effect is not sufficient for understanding its benefit.

2.1.9.2 Precision versus Accuracy (Uncertainty)

Uncertainty elicitation theory suggests that there is a precise value that uniquely represents a person’s knowledge about an uncertain quantity [OhOa04], e.g., that the reduction of a business process’ time afforded by software automation might be \( n \) seconds, on average per process instantiation, with a probability of \( p \). The difficulty is deciding on the representational (measurement) precision of \( n \) (e.g., ‘20 seconds’, ‘\([18,22]\) seconds’, ‘\(\geq 20\) seconds’, etc.) and of \( p \) (e.g., 0.6, 0.604, etc.). As \( n \) reduces in precision (the wider and more uncertain \( n \)'s interval), the higher \( p \) can be, and hence the more accurate the estimate will be. As such, if high accuracy is required, then either \( n \) needs to be less precise, or \( p \) decreased. If neither is acceptable for the decision problem, then more or better information as input to the estimate is needed. Indeed, Jorgensen concludes that expert estimation (the dominant method) is mostly a ‘constructive’ process [Jørg04], and so confidence \( (p) \) in an estimate depends more on the amount of effort spent working on it, than on its actual accuracy.

In general use, the terms certainty and confidence are often used synonymously. However, there is an important distinction: When predicting unknown quantities, uncertainty refers to beliefs about possible values for the unknown quantity (somewhat akin to \( n \) in the previous paragraph), while confidence refers to the belief that a given predicted value is correct \( (p \) in the previous paragraph) [PePi88]. Pragmatically speaking, the most important of the concerns for this thesis’ context is the stakeholders’ confidence in the software requirement(s) contributing at least some particular degree of value, rather than the width of the interval of the possible degrees of contributable value.

Uncertainty is split into aleatory (“due to inherent variabilities or randomness in systems”) and epistemic (“due to imperfect knowledge”) [OhOa04]. Imprecision caused by epistemic uncertainty is reducible by obtaining more or better information, e.g., by increasing the sample size of a survey of opinions, furthering the depth of process modelling (to elicit ‘model conditions’ [OBDE06]), or other uncertainty quantification methods [SwPM09]. Conversely, imprecision caused by aleatory uncertainty could only be reduced if the uncertain event (or proposition, to use Bayesian terms) itself is made more conceptually precise. For example, an ‘avg. customer service call taking time’ interval estimate could be made more precise by narrowing the ‘customer service call’ event to ‘customer cancellation request call’). (Given that aleatory uncertainty may be reduced in this way, i.e., by ‘recognising| and specifying| within the model some more conditions’ [OhOa04], the philosophical question is raised of whether all aleatory uncertainty is just lack of knowledge.)

Ultimately, in this thesis’ context, an estimate’s suitable level of precision depends on the consequences of the risk that the software capability would not contribute the estimated value. For example, if a stakeholder desired at least a 20 seconds reduction in a process time (as a first link in a chain of instrumental value), and a software requirement was estimated imprecisely (e.g., between 15 and 25 seconds with 0.8 confidence), then whether or not to improve the estimate’s precision depends on whether the stakeholder would accept the value implications (and opportunity costs) of a
a 15 second reduction, or a 20% chance for worse (averaging in the worst case an EV of 12 seconds\textsuperscript{15}). Understanding this trade-off requires ‘dose-response’ causal knowledge (discussed in 2.1.2.1) for the stakeholder’s more extrinsic values that would be contributed to by such a time reduction, as well as the estimated contributions of the alternatives to the requirement, to understand the opportunity cost.

<table>
<thead>
<tr>
<th>Summary Point #49: Implication for the Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imprecise estimates of X are quick to elicit, but they are less useful since they don’t rule out as many alternative possible values of X (i.e., X’s true value is more likely to be in a wider estimate interval). If X’s estimate is incorrect, what is the impact of the decision that was made based on that estimate? If the impact is significant, then a more precise estimate of X is warranted.</td>
</tr>
</tbody>
</table>

On a similar subject in RE prioritisation research, Karlsson et al. remarked that “because of the time-consumption and complexity for the user, it would be easier and more efficient to use prioritisation techniques with less rich scales”[KaHR06]. However, they could not answer “the even more interesting question about when the ratio [scale] information is worth the extra effort and when ranking of requirements [i.e., ordinal scale] may be enough”. The above discussion provides an answer to this question using a priori reasoning (as in Hume’s Fork [HuHe95]). (Also, for the interested reader, this problem has been studied as determining the ‘Expected Value of Perfect Information (EVPI)’ [LeSB14].)

2.1.9.3 Conceptual Precision

Finally, while quantification of ambiguous statements can make their claims more precise, it cannot remove all forms of ambiguity [Herb10]. In the field of Natural Language Processing, Herbelot provides the interesting example claim that ‘sharks are dangerous’. Regardless of the quantification (e.g., all sharks, 90% of sharks), the meaning of the statement is subject to the reader’s interpretation of the noun ‘sharks’, e.g., whether it includes dead sharks, toothless sharks, the concept of sharks (future) or physical instances (current), and so on. As well as ambiguity on the scope and physicality of nouns, ambiguity often exists on cardinality and abstraction, e.g., ‘the computer must be backed up’ refers to one instance of a ‘computer’, but ‘the computer is one of mankind’s greatest inventions’ refers to the class of all computers. Such ambiguity in natural language is rarely explicitly resolved such that the statement becomes precise and undeniably true in all possible contexts. Rather, statements are often made with the assumption that other parties will internally resolve the ambiguity, and therefore “assumed clarity is only due ... to the sophistication of [the] reader’s language skills” [Herb10, p.11]. Conceptual modelling can address such issues in RE. For example, the UML [SiBu08] has constructs for ‘is-a’ relationships (as in taxonomy building in ontological modelling [Chan01]) and cardinality between classes. In summary, the precision of a causal proposition for a requirement’s value depends as much on its conceptual ambiguity as its quantitative ambiguity. Rather, the two are interlinked, given that conceptual ambiguity entails quantitative (aleatory) uncertainty. (For example, the causal proposition that ‘implementing requirementX’ will ‘increase user satisfaction’ might have high quantitative uncertainty in the extent of increase expected, due to the various subtypes of the ‘user’ class).

<table>
<thead>
<tr>
<th>Summary Point #50: Implication for the Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>When modelling the value of a software requirement, ambiguity in the concepts used to describe the objects of value will lead to irreducible uncertainty that may be detrimental to the usefulness of the causal propositions for decision making.</td>
</tr>
</tbody>
</table>

\textsuperscript{15} The Expected Value of a 15 sec. reduction (worst case in [15,25]) with 0.8 confidence is 12 sec., assuming that a 0 sec. reduction occurred for the other 20% of theoretical repetitions (the ‘worst’ non-negative scenario).
2.2 Evidence of the Problem (extended discussion from Chapter 1’s Motivation)

The Standish Group’s oft-cited 1995 CHAOS survey of over 14,000 software projects was the first to claim that close to half of all implemented functionalities in software were never used by its end-users [Stan95]. The more recent CHAOS reports show no significant change [John03, Stan10, Thes13, p.2], with the latest results indicating a 50%/30%/20% split (rarely or never used, infrequently used, often used). The Standish Group has now summarised these consistent findings as a ‘law’ in software development that “you will always build too much of what you don’t need, and not enough of what you do need” [Stan10], and advises organisations to treat requirements selection and prioritisation more seriously. Doing so has the additional benefit that smaller projects (i.e., comprised of fewer requirements) are 10 times less likely to fail than large projects [Thes13, p.2]

The problem is also evidenced by independent surveys on feature usage for market-driven or Commercial Off The Shelf (COTS) software. For example, McGrenere studied the problem from a usability perspective by surveying Microsoft Word users on their usage of its features (covering a variety of user roles). After aggregating the results at the user visible feature level, McGrenere found that only 27% of the software’s features were ever used [Mcgr00]. Borg et al. more objectively measured feature usage from the HTTP server logs for an online software product [BKOS04, p.4], and found that 70% of the high level feature categories constituted only 4% of the software’s total usage, with 4 features making up 93.4% of the software’s total usage. Additionally, the vast majority of software features that teams believed to be useful at Microsoft and Google turned out not to be after implementation and evaluation (60-90%) [Koha13, p.22]. Finally, numerous researchers have concluded that between a third to a half of a typical software project’s effort is spent on avoidable re-work caused by immature understanding of requirements; In three independent studies, Boehm and Basili concluded on a 40-50% rate [BoHZ05, p.427], Leffingwell concluded on 40% [Leff97], and a more recent IAG Consulting survey concluded on a ‘one third of development effort’ rate [Ruff09].

Evidence for the existence of wasteful non-functional requirements also exists, although mostly for wasted opportunity (i.e., inadequate quality), as opposed to wasted implementation, (i.e., superfluous quality). Examples include that Google recently concluded that a 500ms response time delay in their search product would lead to a 20% revenue loss, while Amazon attributed a sales decrease of 1% for a 100ms response time delay in their e-commerce platform [MaKh14]. Similarly, Microsoft concluded that reducing response time by a mere 10 milliseconds in their Bing search product “more than pays for [an engineer’s] fully loaded annual costs” [Koha13]. The reader might argue that slow response times were unlikely to be prescribed as acceptable quality levels in requirements for these products, and hence that they concern implementation inadequacies, rather than requirements inadequacies. However, the quoted statistics are retrospective rather than predictive, in that the negative consequences (namely loss of revenue or potential for it) did occur, suggesting that the requirements were either insufficiently prioritised or inadequately specified. (I.e., even if these problems were mostly caused by implementation inadequacies, it is unlikely that RE had no role to play.) Overall, there is less evidence of wasteful NFRs than FRs, perhaps because quality cannot be ‘used’, and so waste is less visible, or because non-architectural NFRs are operationalised by FRs [Paec04].

Finally, there is significant evidence to support the claim that opportunity for valuable software requirements is wasted. Numerous studies (aside from the CHAOS report mentioned in Chapter 1) conclude that less than half of all software projects realise their expected benefits in the majority of organisations; For example, Nelson claims that only 30% of projects actually deliver their expected benefits [Nels07]. Panorama consulting’s survey of over 2000 Enterprise Resource Planning
(ERP) software implementations from 2010-2014 show that 66% of the respondent organisations failed to realise at least half of their project’s expected benefits, while 72% exceeded their schedule, and 54% exceeded their budget [Pano10]. Similarly, Akhavan et al. claims that 50-70% of Knowledge Management IT projects fail to achieve their expected benefits [AkJF12], with the blame most often being placed on solution rather than problem orientation, e.g., “just moving data around may or may not add value to anyone in the enterprise” [Ambr00], and on “failure to align ... with the organization’s strategic objectives” [FoLe02]. (The reader might interpret this evidence to support the theory that benefit realisation failures are due to optimistic expectations, rather than blaming software or requirements engineering. However, this is still a type of value realisation failure, and is treated by this thesis.) Furthermore, nearly a third of all Chief Information Officers in the USA were not confident that their IT investments had generated a good return for their organisation [PeWD07]. More pessimistically, only 15% of executives surveyed by Forrester believed their technology projects to be fully aligned with business goals [Forr07], despite that numerous surveys such as Accenture’s concluded that a major project success factor is alignment of IT and business goals [Acce04]. All of this makes it clear to see why the “top ranking concern” of business executives for the last two decades has been IT-Business alignment [DeLu14, JlTb11].

2.2.1 Where did this evidence come from?

The CHAOS reports have risen to be one of, if not the most cited industry benchmark in IT [EvVe10], and are a primary source of evidence to support ‘the software development crisis’ [Carr04]. However, the CHAOS research methodology and results have been criticised by researchers. Firstly, “Standish hasn’t explained, for instance, how it chose the organizations it surveyed, what survey questions it asked, or how many good responses it received” [EvVe10]. This may be common practice in commercial research (since the data is Standish’s competitive advantage), but it is now well accepted that where researchers keep data, sources and methods hidden, then low reliability should be placed on the conclusions [Zveg98]. Secondly, Jørgensen et al. highlight that the CHAOS report is geared towards proving software development failure [JøMo06], since the CHAOS definition of project success discards project under-runs and so exaggerates over-runs. To further support their argument, an excerpt from one of Standish’s documents is cited: “we then called and mailed a number of confidential surveys to a random sample of top IT executives, asking them to share failure stories” [JøMo06]. Hence there is the risk that CHAOS’ conclusion on wasted functionality (and success) is biased. In response to these criticisms, Standish replied that “All data and information in the CHAOS reports and all Standish reports should be considered Standish opinion and the reader bears all risk in the use of this opinion” [EvVe10, p.36] — a worrying disclaimer for the robustness of their results. Thirdly and finally, both Sommerville [Somm11b] and Glass [Glas06] reject the notion of there being a crisis in software development (given common comparisons to more successful development projects, such as bridges [Atwo05]), due to the existence of the ‘computer age’. Jørgensen et al. view the CHAOS results from an alternative perspective [JøMo06]: perhaps the results merely indicate a software development estimation crisis, given that whether a project is successful or not depends on the gap between actual results and predicted results (predicted cost, time, and amount of functionality included). In a more recent paper, Jørgensen finds that software (effort) estimation accuracy has seen no significant improvement since the 1980s [Jørg14]. (E.g., advances in statistical models have not been effective, tending to “overfit to the historical data and will consequently be less accurate than simpler models when the context changes” [Jørg14].) This viewpoint does not affect the interpretation of the claim that ‘too many non-useful functionalities are being
built, while not enough useful functionalities are', since requirements are essentially predictions that a software capability would be useful to some (perhaps implicitly and qualitatively) estimated degree.

More relevant to this thesis, is that there are terminology inconsistencies in the CHAOS literature on the usefulness of implemented functionality: The 2010 report claimed that "50 percent of software features are not used or wasted" [Stan10], whereas the 2013 report claimed that "50% of features are hardly ever or never used" [Thes13]. The latter’s use of “hardly ever” makes the claim less impactful than the former’s due to inclusion of features with non-zero usage, but more important is that the inconsistent reporting reduces comparability, and also the overall trustworthiness. It is also plausible to be concerned about the robustness of the results, given that they are based on IT project member’s perception of the degree of unused functionality in their software project; It is not based on an objective empirical measurement, and it is unlikely that the participants ‘properly’ calculated their response by mentally enumerating each item of functionality. It is also logically possible for two participants to claim 50% usage of one project’s functionality, meaning that 100% of the software’s functionality was used, in contrast to the response average of 50%. However, such an occurrence is highly unlikely, and the robustness of the results should have increased in accuracy over time with the growth of web-based applications and hence the availability of usage statistics.

Interestingly, A KPMG survey reported that only 11% of the respondents’ IT projects had performed benefits realisation analysis [Kpmg08], yet far more provided their opinion on whether their IT projects were successful. Similarly, Eckartz found that most practitioners (64%) did not know the percentage of their software projects’ planned benefits were actually realised [Ecka12]. This exemplifies two points: Firstly, opinions on the success or failure of software are mostly founded on ‘gut feeling’ rather than empirical data, and secondly, the IT/software industry does not care much for understanding the actual benefit provided by IT. (Further supporting the claim that software projects tend to be managed in a ‘value neutral’ setting, where cost and schedule [Boeh05] are monitored rather than value delivered to the stakeholders.) Indeed, research and practice on estimation in the software engineering community has focused almost entirely on estimating the effort (cost) of developing software [Jørg04], perhaps due to the fixation with the ‘cost to fix curve’. (Nevertheless, it is not impossible to infer knowledge about benefit realisation from the myriad of conclusions on software project time and cost overruns, because when a requirement’s implementation is not considered adequate, more effort, resources and time are likely to be spent in re-development or reparative work.)

Regardless of the above concerns, and reassuringly for the robustness of the surveys discussed in this section, Borg et al. found 90% agreement between measured use of features elicited via analysis of HTTP logs, and the stakeholders’ perceived use of the features elicited via survey [BKOS04, p.8]. Furthermore, an independent academic survey in 2007 set out to replicate the CHAOS ‘software crisis’ finding, found that the software project failure rate was between 17% and 22% [ElKo08, p.88], which was in line with the closest CHAOS report’s (2006) 19% finding [ElKo08, p.85]. Overall, the evidence discussed in this section is believable. The main problem from this author’s point-of-view is its relatively scarcity, given the claimed extent of the problem, in addition to that the reports tend not to describe the context of the results (e.g., breaking down results by industry, type of project, country, etc.), nor explore the causes beyond vaguely blaming ‘poor or inadequate RE’.

Finally, Johnson and Holloway “urge greater caution in identifying requirements failure as a ‘catch-all’ cause of adverse events” [JoHo06], especially when considering claims such as that “over 90% of safety-critical systems failures are requirements errors” [Ladk98] or that RE is the chief cause of software project failure (as discussed in Chapter 1). They argue that just because RE techniques could have avoided software project failures does not mean to say that they would have, and that
claims of RE’s liability are likely to be affected by hindsight bias and logical fallacies such as *dicto
simpliciter* (e.g., ‘studies show that RE is the #1 cause, so it must be in this project too’),
*argumentum ad ignorantium* (e.g., ‘RE is the cause since there is no evidence to suggest that RE
was adequate’), and *post hoc ergo propter hoc* (e.g., ‘root cause analysis traces to early events, RE
is early, therefore RE is the cause’). In conclusion, and to paraphrase Brooks [Broo87], some
proportion of RE errors leading to wasteful implementations will be avoidable (‘accidents’), and the
other proportion will be unavoidable (‘essential’); It is inevitable that some proportion of wasteful
implementations (or project failures) attributed to inadequate RE may in fact be due to non-RE
factors such as poor project management (as Jones argues [Jone95]), or are simply part of the
‘unavoidable essence’ of software engineering. Hence, this thesis’ framework cannot be expected to
resolve all of the reported problems that it is motivated by, — just as any other RE technique cannot.

2.2.2 (Theories on the) Causes of the problem

Jain & Boehm’s theory of Value Based Software Engineering offers two explanations for the (perhaps
pessimistically phrased) “dismal state of affairs in software systems development”[Jain07] (purportedly
indicated by low success rates [Thes13], loss of revenue [Carr04], and even loss of life [FiDo96]):

a) The scope of software projects is overly limited both before and after implementation. For
example, acceptance tests are often interpreted as checks for “*certain [system] behaviour
and/or output*”[MiCo01], rather than checking for adequacy against stakeholder problems.

b) Project progress is tracked by timely completion of functionality rather than delivery of
stakeholder value. Trade-offs between competing stakeholder desires are rarely considered,
and more rarely guided by processes. Software project members are infrequently held
responsible for value delivery, and rarely take responsibility for non-software initiatives often
required to realise the benefits associated with the software (as also argued in [PeWD07]).

Ambler offers four explanations for wasteful software features being implemented [Ambl05]:

a) Insisting that stakeholders define requirements up front will lead to the specification (and
prioritisation) of features with the mentality of “I don’t know if I’ll actually need this feature,
but if I don’t ask for it now, I may never get it”. Description of rationale could avoid this.

b) Specified features will become obsolete as the environment (and hence desires that were
once valued by stakeholders) evolves. Note that more extrinsic values are less dynamic, and
so through explicit modelling of instrumental value chains, obsolete links can be identified.

c) Change management processes imposed by the software vendor or purchasing organisation are
prohibitive of change initiated by changes in environment or as a result of requirements analysis.

d) People are better at stating what they don’t want rather than what they do want (the IKIWISI
problem [Boeh00], or in other words, errors of omission are less visible than of commission).
However, this is more applicable to solution requirements rather than problem requirements.

Overall, there are a myriad of possible causes for implemented software capabilities being redundant
(wasteful), classifiable along the who, what, where, when, why dimensions. As briefly mentioned in the
previous section (2.2.1), not all of these causes are due to misalignment between the software capability
and its expected benefits. For example, a piece of software functionality could have been useful,
except for that it was delivered late (‘*when*’), or that it would benefit the organisation but not the
un-motivated user (‘*who*’), or that a competing software product was favoured due to politics rather
than value (‘*what*’), and so on. Due to the possible combinatorial complexity of these causes,
constructing an exhaustive taxonomy would not be practical. This thesis’ interest is primarily in
improving the RE process such that the factors necessary for value realisation (and hence factors

58 2.2 Evidence of the Problem (extended discussion from Chapter 1’s Motivation)
sufficient for value failure) are risk reduced or mitigated. Nevertheless, results of investigations into the causes of wasteful implemented FRs or NFRs are presented later in Chapter 6’s Case Studies.

2.3 Context of the Problem

In a paper titled “Matching Methodology to Problem Domain”, Glass notes that software engineering researchers often propose methodologies without explaining where they would, and would not be useful [Glas04]. Universal applicability can so rarely be assumed, and argument often ensues when it is claimed, e.g., in the applicability of UML in device driver programming [Spol02], or Agile techniques in large organisations [Buck10], outsourcing [Amb105], or safety critical software [CIGZ12, p.vi]. Claims about the pros and cons of any solution depend on its environment of use (e.g., a 20MB runtime required for a software framework cannot be claimed to be a ‘non issue’, since it clearly is when developing for a device with a 32KB ROM [Spol02]). Hence this section describes the environment in which the thesis’ problem exists, and where its proposed framework is most applicable.

2.3.1 When

Regnell et al. propose a decision oriented model of a requirement’s lifecycle, whereby a requirement is “decided to be elevated or downgraded individually in a continuous, asynchronous refinement process” [RPAW01]. In order to contextualise when this thesis’ problem-of-interest occurs, and hence when the framework is hoped to be useful, Regnell et al.’s lifecycle diagram is annotated in Figure 2.17. (The traditional ‘cost’ bias (one of the ‘iron triangle’ project success indicators) is visible in Regnell et al.’s lifecycle, since a requirement is “cost estimated” between [C] and [D], but is not “benefit estimated”.)

Figure 2.17 makes it apparent that RE process improvements are not limited to improved specification [B] — the traditionally considered ‘output’ of RE [Ecka12, p.120]. Rather, the quality of each of the stages and their transitions can be improved, which may be intangible, e.g., in stakeholder confidence. (Interestingly, confidence in a requirement is desired earlier in the lifecycle [SyGe13] due to the ‘cost to fix curve’ [BoBa01], but the ‘cone of uncertainty’ [Boeh81, p.311] dictates that the opposite is possible.)

Specifically, the following RE decisions and lifecycle transitions annotated in Figure 2.17 are expected to be aided by this thesis’ framework:

---

Figure 2.17: Lifecycle of a requirement as proposed by Regnell et al. [RPAW01]. The stages where this thesis’ framework is most applicable are annotated with grey labels in square brackets (e.g., [A]).
A. Elicitation of new Candidate Requirements

The phrase ‘modelling the value of a software requirement’ implies that value analysis occurs ‘bottom up’ from an existing requirement. However, understanding stakeholder values can lead to new requirements, as well as stakeholders who could benefit, disbenefit, or be responsible for them. Furthermore, the framework’s software tool can recommend benefits/obstacles/requirements, etc.

B. Approval of (or Discardation of) a Candidate Requirement

Both stakeholder and system requirements are solutions to a higher level problem. Modelling the estimated contribution of a requirement to that higher level problem helps to understand, communicate, provide rationale, and receive feedback on the decision to commit resources to the requirement. Hence, the framework is expected to aide ‘feasibility studies’ of the requirement, and hence of the project (defined as an early evaluation of its value, viability, & integrate-ability [Somn11a, p.100]). (The feasibility phase of a requirement has traditionally received little research [Fink94].)

C. Specification of an Approved Requirement

Modelling the mechanism by which value is to be created by a software capability involves modelling the causal necessities for value. Hence it is a value oriented means of specifying requirements. In order to specify optimal requirements, the framework is expected to support impact analysis (how do changes to a requirement propagate up to affect the stakeholder values?), trade-off analysis (which requirement in a state of conflict with another is more important?), and non-functional requirement sensitivity analysis (how far can a requirement can be ‘relaxed’?, e.g., exploring dose-response effects of system uptime from 99.99% to 98% availability per week on stakeholder values).

D. Release Planning of a Specified Requirement

Understanding how valuable a requirement is, has useful implications for prioritising requirements, (deciding on the order of implementation and the likelihood of being discarded with budget cuts). As such, value analysis has obvious and empirically validated applicability to the release planning activity (the packaging of requirements into versions of the software to be released to users). Furthermore, given that this stage [D] follows specification but precedes implementation, it includes decisions about alternative requirements (1), and predictive validation of requirements (2):

1. There is a risk that the criteria used (typically stakeholder desires) to evaluate alternative solution requirements will themselves not be as valuable as expected. This risk could be reduced (and hence confidence in them can be increased) by treating those criteria as instrumentally valuable, so that their alternatives can be considered.

2. Confidence that the specified requirements will sufficiently satisfy the stakeholders after their implementation can be gained by the validation of requirements against higher level desires. This thesis’ framework is clearly and directly pertinent to this RE activity.

E. Verification & Validation of an Implemented Requirement

Although Figure 2.17 makes no reference to the requirements validation activity [Clei05], this thesis’ framework is more pertinent to it than the verification activity. According to the engineering ‘V’ process model [FoMo95], retrospective evaluation of a requirement after implementation is required in order to determine its adequacy, and ultimately the project’s success. This thesis’ framework is expected to help elicit the criteria by which a stakeholder would consider an implemented requirement to be successful, and to assist the re-use of this information in future predictive validations.

---

16 In a survey on the influences in requirement selection in software product releases, Wohlin & Aurum show that four criteria dominate the thirteen identified: cost-benefit; assigned priority; impact on delivery date; and who the originating stakeholder is [WoAu06]. More interestingly, when asked what should be the dominating criteria, practitioners believed that the importance of cost-benefit should be increased.
There are two primary concerns on the suitability of this thesis’ framework to an organisation:

1. Is the organisation in a position to (i.e., willing and able) use the framework?
2. How beneficial would the framework be in an organisation’s software project?

To understand the first concern, the maturity of the organisation should be evaluated, with respect to its capabilities, employee motivation, use of best-practice, and so on. (Numerous RE capability maturity models exist for such an assessment [ODKE05, Sei12, SoRa05, SoSa97a]17.) An immature organisation is less likely to be interested in RE, and even less in a comprehensive RE framework such as this thesis’; “Practitioners often seem reluctant to spend some effort in the RE process. They are in a sense, like cigarette smokers who know that smoking is pretty unhealthy but keep smoking” [Lams08]. Indeed, “Requirements Engineering rarely receives more than 2-4% of the overall software project effort” [Fire04], despite evidence showing that increasing effort spent on RE improves the probability of the project’s success [AbPa13, AMBD04, DaCh06, MaTh13a]. Conversely, a highly mature organisation that spends more effort on RE may be too heavily invested in their own processes, and so certain elements of the framework may be adopted or influence changes to existing processes, rather than the whole framework being adopted.

The second concern relates to organisational characteristics and trends (project characteristics are discussed in the next Section 2.3.2). Specifically, organisations exhibiting the following symptoms (elicited from primary as well as secondary sources [FoCa01, TrHS11]) are likely to gain the most from the application of this thesis’ framework:

- The value of an IT or software is unclear amongst stakeholders or management, hence funding, management buy-in, or stakeholder involvement in a software project is difficult.
- Evaluation of the success and failure of IT projects or business strategies is unclear.
- IT and software development teams are:
  - seen as ‘cost drivers’ that are easy to replace, and play no significant role in creating advantage for the organisation;
  - confronted with unrealistic goals or perceptions of software being ‘plug and play’;
  - outsourced or distant from stakeholders, resulting in gaps in expectations, understanding of needs, and responsibility for delivering more than ‘quality’ software;
  - interacting with interdisciplinary stakeholders, or those unfamiliar with the solution domain;
  - dynamic, in the sense that team members are frequently leaving and joining.
- Business stakeholders are unsatisfied with the outcome of IT or software projects (specifically, its capabilities and qualities, timing, and the extent of benefits and disbenefits).
- Organisational budget constraints have mandated rigorous business cases for investment, especially where a software project’s sponsors are likely to be outranked in political influence.
- Requirements specifications are long, rigorous on principle, rarely read by developers, text-based (long lists of ‘system shall…’), solution-oriented, and don’t describe what must change.

2.3.3 Who

Requirements engineering is a multi-disciplinary activity, requiring knowledge of constructing software components, as well as knowledge of stakeholder problems and business processes. The

17 The capabilities in Olsson’s RE capability maturity model [ODKE05] of most interest to this thesis’ framework are: Plan Releases, Estimate Time, Document Alternatives, Introduce Traceability, Document Rationales, Elicit Goals, Model Requirements, and Describe Testable Requirements.
Implementation of a methodological approach or framework tends to be managed by a designated facilitator, and this framework is no exception. An understanding of the concepts described in this section is considered necessary, along with the typical qualities for people-facing roles (communication, leadership, conflict resolution, etc. [RoRo12a]). Stakeholder participation, is of course crucial: In an argument about the value of a requirement, the default position is (or rather, should be) suspended belief, i.e., neither for nor against its worthiness for implementation, unless there is some reason to believe it. The responsibility for providing evidence and reasoning for the claim that the requirement is valuable (i.e., the burden of proof [BrSt01, p.117]) lies with those seeking to persuade holders of the default position. In other words, the burden of proof lies with the stakeholder requesting a requirement, or the requirements engineer proposing an alternative to it, rather than on those who might reject the requirement (i.e., the ‘gatekeeper’ in a requirements release planning or quality reviews [MaMa03]). This is logical in the same way that the burden of proof should lie with someone who makes claims that are difficult to falsify or verify, e.g., that a teapot orbits the Sun (as humorously proposed by Bertrand Russell) [Lowe11, p.65].

In cases where the software is to be developed and used by different organisations, then it should be the responsibility of the organisation who is acquiring the software to ensure value, and hence use this thesis’ framework. Nevertheless, some (better quality) software vendors who realise that satisfying their customers is more important than delivering software in contractual compliance with the approved specification. In other words, good software architects do not trust that the requirements engineers did a good job [ClBa10], and so may be interested in this thesis’ framework for validating the requirements before the design work starts.

2.3.4 (For) What

The type of software product for which requirements are to be ‘engineered’ has implications for the suitability of this thesis’ framework.

Client vs. Market: This thesis is more focused on client oriented than market oriented software, e.g., the specification for an information system for a particular business, rather than the a new word processing package for the population [Wier96]. Understanding consumer needs, and hence desired value, is far less ‘fuzzy’ in client oriented software, because the need that starts the development process is experienced by the client, and so is more likely to exist. In market oriented development, needs are less definite, are perceived by marketing specialists, and are more generic since solutions are not bespoke to a particular context. While this thesis’ framework could apply to market oriented software, most causal propositions for benefit would inherently be more aleatory uncertain and less precise (conceptually and quantitatively) in order to be applicable in more cases (covering a wider range of consumer profiles, use cases, and environments).

Development and Acquisition: This thesis is primarily interested in the RE activity, regardless of whether the prescribed capabilities require either software acquisition or development. However, when market driven software (primarily Commercial Off The Shelf (COTS) software) is to be acquired or customised, the project is client oriented and hence overrules the previous paragraph.

Software Type: Spolsky proposes five categories of software product (listed in order of applicability to this thesis): Internal (to be used in a particular organisation’s IT infrastructure, e.g., a trading platform for a particular bank), Shrinkwrap (commercial or open source, e.g., MS Word or OpenOffice),
**Embedded** (e.g., in-car GPS), **Games** (e.g., Duke Nukem 3D), and **Throwaway** (e.g., shell scripts) [Spol02]. Each of these types of software has different quality trade-offs and implications of RE failure, e.g., a requirements error might be more costly in embedded systems if updating the software is more difficult, or, gathering requirements from users for internal software (fewer users) might be easier than for shrinkwrap. In summary, this thesis is primarily interested in sociotechnical systems, i.e., those including technical (e.g., software) as well as nontechnical (e.g., human) elements [Somm11a, p.267], given that there is greater scope for stakeholder value analysis in systems directly involving stakeholders — especially systems that change the future behaviour (e.g., processes) of humans.

**Online vs Internal (or Offline) Infrastructure:** A software product to be deployed as an internet web application is more easily changed than software that is traditionally ‘installed’ on a user’s computer. Arguably it is more important to be ‘right first time’ in the latter case (especially in embedded software), and so the importance of value based requirements analysis is increased. This importance is further increased where design often preceedes analysis of needs, such as for web applications, having inherent constraints on architecture and user experience, hence and standard solution patterns [ZoGe01].

**Large vs. Small Software Project:** A small software product (e.g., one developer following a build-and-fix development methodology for a personal project) requires less rigour than a larger product. It would clearly not make sense to have a better quality requirements specification than software product, especially if the requirements could have more easily been communicated vocally (i.e., in tightly-knit teams, there is less need for explicit RE models [Somm07a]). Large projects are inherently riker (as empirically validated [Thes13]) than smaller projects, having considerably larger and more complex stakeholder social networks [ReSW08], and hence more (often conflicting) expected benefits to manage. Overall, projects suitable for the framework should meet at least the criteria for ‘Medium-Scale RE’ [ReSW08] (~100 requirements, with numerous stakeholder classes).

**Agile vs. ‘Traditional’ Project:** Teams adopting an agile approach to software development often erroneously argue that RE (and documentation in general) is incompatible [PaEM03], and should be eschewed [CIGZ12, p.vi]. This thesis’ framework is independent of development process, in that it does not stipulate when development should start and when requirements engineering should end (although guidance is offered). The ‘Big Requirements Up Front’ development process is often required by law, or for compliance with standards (e.g., in Defence or Aerospace) [SBJA98], and projects that do not follow the Agile methodology are not always so na"ive as to believe that software releases should be delivered to users infrequently, or that RE is a ‘one-shot process’. (E.g., Pahl et al. warn that “Any attempt to formulate all possible requirements at the start ... would cause considerable delays”[PBFG06]). However, an Agile proponent could argue (and has [Vanc09]) that requirements validation is meritless compared to rapid feedback of early iterations of software functionality. Yet, amongst numerous often overlooked assumptions [Lams09a, p.54], this presumes that end-users will be available to frequently provide feedback, and that prototypes or early-versions are relatively affordable to produce. (The logic behind RE is that early identification of incorrect assumptions tends to be cheaper using requirements, or at least prototypes, rather than by just building the software (‘build and fix’), especially in complex domains such as engineering process automation compared to the more typical Agile project of producing a web CRUD form in HTML). *(CRUD: ‘Create Read Update Delete’ capabilities, generic to most business applications [MaVN06, p.40].)*

---

18 The founders of Agile did not advocate skipping RE. Indeed, ‘User Stories’ are central to it [Cohn05].
2.4 Summary

To summarise, a software requirement has been described as means to an end, where an end is expected to be beneficial, after the costs of implementing the means and the disbenefits of the end are considered. It is in this sense that a software requirement is considered to be instrumentally valuable, i.e., beneficial for other benefits, which themselves are beneficial for other benefits. The other concepts illustrated in the conceptual framework (Figure 2.1) are those considered to be useful when building a model of a software requirement’s instrumental value.

Finally, this chapter has shown (by describing requirements as uncertain strategies for achieving a higher desire), that this thesis’ abstract problem is relevant to wider fields than software engineering. The retailer F.W. Woolworth is claimed to have said that half of his advertising costs were wasted, — he just wished he knew which half [Stan10]. This analogy is apt, since:

1. Knowledge of the advertisement’s (or software capability’s) benefit is valuable for resource allocation optimisation. However, it must be re-used and re-examined to avoid repeated mistakes. (This is to be achieved by making models of value explicit, codified, and retrievable.)
2. A priori knowledge of the predicted causal relationships leading to benefit will always be uncertain due to sociotechnical environments being non-deterministic and highly variable. (This is to be achieved by communicating uncertainty and confidence in predicted value.)
3. A posteriori knowledge of value (e.g., by evaluation) to be gained after the advertisement or software requirement has been implemented takes effort to be known, but is necessary to gain more than superficial knowledge. (This is to be achieved by ‘post-mortem’ of projects, and support for historical tracking of a software capability’s expected and actual value.)

This chapter has fully answered research questions 1.1 (“What is the motivation for predictively modelling the value of software requirements”) and 2.1 (“What is the conceptual framework…?”). It has partly answered questions 2.2 (“What are the causes of software value realisation failure?”), 2.3 (“Which RE decisions could be made differently to prevent the value failures?”), and 2.4 (“What features would be useful in such an approach?”). The next chapter will evaluate potential research methods, and provide rationale for those chosen for this thesis.
3. Research Methodology

“Adventure is just bad planning.”
Roald Amundsen
The subject of this thesis lies at the intersection between the fields of Software Engineering (SE) (the production and operationalization of software) and Information Systems (IS) (the deployment of Information Technology (IT) in an organisation [HeCh10]). While the research methodologies from these fields might vary in terminology or process, their concerns and outcomes ought to be similar, given that:

i. Information Systems research is interested in the synergy of technical and social systems together [VaKu07], and;

ii. Ultimately, all software exists only to have effects on its application domain [Jack95a] (which in this thesis’ context, is an organisational or other sociotechnical environment).

Therefore, research methodologies, philosophies, and strategies from both Software Engineering and Information Systems are considered to be applicable to this thesis. This might be considered abnormal, given that Requirements Engineering (RE) is often categorised as a sub-discipline of Software Engineering (e.g., as node [D.2.1] in ACM’s taxonomy), — but Requirements Engineering is irrefutably as applicable to the acquisition of software as it is to the engineering of it [NeMa99].

This chapter explores the potentially applicable research methodologies, and then concludes on the appropriate research philosophy (Section 3.1), research strategy (Section 3.2), and research design, method and process (Section 3.3).
3.1 Research Philosophy

According to Myers, the two main and contrasting research philosophies associated with epistemology (the nature, scope, and means of acquiring knowledge) are *Positivism* and *Interpretivism* [Myer97]. However, Creswell identifies four dominant research philosophies [Cres09], adding *Critical Theory*, and *Pragmatism* to the list, and reassuringly, the same four philosophies are identified for the software engineering research context by Easterbrook [ESSD07]: *Positivism, Interpretivism, Critical Theory, and Pragmatism*. (The importance of choosing a research philosophy is summarised with the question “what will you accept as valid truth?” [East06, p.8].)

**Positivism** postulates that all knowledge must be based on logical inference from basic observable facts. Positivist research is a form of reductionism, in that it studies phenomena through decomposition into its simpler components. Correspondingly, positivists believe that knowledge is constructed from verifiable observations of external reality and the inferences based on them. In other words, positivism “generally assumes that reality is objectively given” [Myer97]. Positivists generally start research with clear-cut theories from which verifiable hypotheses can be generated, and then tested in isolation. As such, positivist research favours controlled experiments and quantitative approaches [East06, p.8], as are popular in the natural sciences (hence positivism is also known as the ‘scientific research’ philosophy [Myer97]). Less controlled methods such as questionnaires could also be used in accordance with the positivist philosophy. E.g., survey questions could be refined from hypotheses, and then statistically significant conclusions could then be inferred from random samples of the population. Table 3.1 summarises the pros and cons of Positivism in the context of this thesis.

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
</table>
| - Provides a ‘top down’ structure for decomposing and answering research questions in a rigorous scientific manner, due to its reductionist viewpoint;  
- Provides conclusions with transparent trustworthiness (e.g., confidence levels in statistical inference), due to its objective and usually quantitative nature;  
- Provides controllability and the ability to plan the research process [Radd10];  
- Enables a clear focus for the research from the outset, which may be useful in finding funding or cooperation;  
- Good for generalising findings in order to verify or falsify theories [East06], as opposed to exploring the findings and their context;  
- Favours quantitative data avoids long subjective texts. | - Can be limited in applicability for studying non-deterministic phenomenon such as sociotechnical systems, where experimental repeatability with exactly the same conditions is not possible [ESSD07];  
- Conclusions can be non-holistic, since reductionism’s decomposition of phenomena into components ignores their composition (synergy is ignored);  
- Lateral thinking can be discouraged if there is faith in ‘objective’ results, e.g., confidence levels such as the standard 95% [Karl02] might be interpreted as 100%;  
- Complete objectivity in human activities is not always possible, and aspiring for it may provide diminishing returns (humans are not always rational and are affected by biases, ulterior motives, etc.). |

**Interpretivism** opposes positivism (and so it is also known as anti-positivism) since it states that researchers can obtain different results from the same studies, due to different possible interpretations of the data and depending on the context of the situation. Specifically, it rejects the idea that scientific knowledge can be separated from social (i.e., human) context [KIMy99] and seeks to explore subjective meanings. As such, interpretivists concentrate on understanding how different people interpret the studied phenomenon rather than objectively verifying theories about it. Interpretivist theories tend to be developed ‘bottom-up’ from data to conclusions, hence Interpretivism is also known as constructivism. Interpretivist research prefers methods that collect qualitative data about human activities, such as ethnographic studies, exploratory case studies, interviews, and survey research. Table 3.2 summarises the pros and cons of Interpretivism.
Table 3.2: Pros & Cons of the ‘Interpretivism’ Research Philosophy

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Conclusions about ‘why?’ phenomenon (especially social) occur are more easily provided than with positivist research, thanks to qualitative data which tends to be rich in context and depth of description;</td>
<td>• A focus on qualitative rather than quantitative research can undermine data representativeness and reliability, since it is more open to interpretation and more easily affected by cognitive biases;</td>
</tr>
<tr>
<td>• Unexpected conclusions are more likely than in positivist research, since hypothesis tend to be formed after, rather than before data collection;</td>
<td>• Generalisation of results (external validity) can sometimes be difficult, since results tend to be highly context specific and personal [RuHö08].</td>
</tr>
<tr>
<td>• Good for generating theories [East06], since patterns, themes, and holistic features are the focus of data analysis in interpretivist research.</td>
<td></td>
</tr>
</tbody>
</table>

Critical Theory evaluates scientific knowledge by its ability to “free people from restrictive systems of thought” [Calh95]. Critical theorists tend to involve the stakeholders that the researchers are trying to help, especially when setting the goals of the research. Hence, some consider critical theory research to be political acts that empower certain groups or individuals [East06], perhaps at the expense of other (e.g., those supporting the ‘restrictive systems of thought’). Perhaps the most notable critical theory research in Software Engineering are focused on the open source and Agile movements [ESSD07]. In practice, critical theorists often use Case Studies to draw attention to phenomena requiring change, and then Action Research to evaluate the changes. (Finally, it is important to clarify that ‘critical theory’ is distinct from ‘critical research’, which is to “not claim more than [the researcher] can justify, and to this end reflects on all possible ways in which [the researcher] could be wrong” [WiHR09]). Table 3.3 summarises the pros and cons of Critical Theory.

Table 3.3: Pros & Cons of the ‘Critical Theory’ Research Philosophy

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Good for making a quick impact on practice or society (relative to positivism or interpretivism), due to high stakeholder involvement in defining the research questions [East06];</td>
<td>• Research may be affected by selection bias (where research that is not in agreement with ideologies is downplayed), since research questions are set depending on ‘who it will help’ rather than on ‘what is important to research’;</td>
</tr>
<tr>
<td>• As a consequence of the above, research outputs tend to have high practical relevance, which in turn can increase research output due to the feedback loop between practice and theory.</td>
<td>• Society may not have stakeholders interested in some important research questions, causing either neglected research topics, or critical theory’s participatory and political nature to be redundant.</td>
</tr>
</tbody>
</table>

Pragmatism adopts a problem solving approach to research, and so is popular in engineering. It values practical knowledge over abstract knowledge (i.e., “truth is what works at the time” [East06]), and leaves the researcher free to use any method that they consider to be most suitable for answering the research question. Pragmatism acknowledges that all knowledge is incomplete and all forms of inquiry are biased [East06], but that some is more so than others. Therefore, the usefulness of some piece of knowledge depends on the problem it solves, as well as how it came to be known (e.g., field experiments are preferable to opinions). Crucially, pragmatism recognises that there is a cost involved in more robust research methods, and so is willing to trade-off robustness for effectiveness and efficiency. Pragmatists therefore tend to prefer mixed methods research (i.e., where several methods are used), since the various methods will be more or less suitable for various research questions. More recently, the pragmatic ‘Design Science’ paradigm has become popular in both Information Systems and Software Engineering [Wier09a, WiHe08] due to its focus on solving practical problems. Table 3.4 summarises the pros and cons of Pragmatism.
Table 3.4: Pros & Cons of the ‘Pragmatist’ Research Philosophy

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Allows flexibility in research method selection, which helps prevent “getting hung up on methodological purity”, since “some knowledge is better than none” [East06];</td>
<td>• The robustness of the research output may be criticised since the focus of the research is not on the ‘truth’ of the results;</td>
</tr>
<tr>
<td>• Practical problems are more likely to be solved with better cost effectiveness than with the other philosophies, since the research results’ robustness, fidelity, integrity, richness, and socio-political power come second place to its usefulness.</td>
<td>• Time pressures and focus on solving specific and perhaps context-specific problems can lead to inaccurate and non-generalisable conclusions, or worse, derailment of the research project to become mostly a practical problem solving project.</td>
</tr>
</tbody>
</table>

3.1.1 Comparing the Rigour of the Research Philosophies

In 2004, Dawson et al. asked “what constitutes evidence?” in software engineering research, arguing that “it is difficult to prove anything” with interpretivist research and that while positivism could provide the ‘fundamental truths’, “it is difficult, perhaps impossible, to perform in a software engineering context” [DBOB03]. Such difficulties have long been felt. For example, more than two decades ago, Checkland stated that “If the assertion is: ‘The methodology does not work’, the author can reply, ungraciously but with logic, ‘How do you know the poor results were not due simply to your incompetence in using the methodology?’” [ChSc90, p.299].

To answer their “what constitutes evidence?” question, Dawson et al. constructed a ‘pyramid of empirical research types’ for software engineering (Figure 3.1), where the tip of the pyramid represents the ‘rarely achieved’ positivist research, and the base of the pyramid ‘firmly placed on the ground’ represents the more common interpretivist research type. (Kitchenham’s list of evaluation methods ordered by their associated cost [Kitc96, p.14] follows the same order as Dawson et al.’s pyramid, which adds to its credibility).

Figure 3.1: Pyramid of empirical research types in software engineering [DBOB03]

More recently on the topic, Wieringa provided an ordered list of validation methods to be used in design science [Wier09b], where the order of the validation methods is based on objectively assessed variables, such as whether the researcher has control of the context:

1. Case study
2. Pilot project
3. Action case
4. Field experiment
5. Field trial
6. Benchmark
7. Lab experiment
8. Lab demonstration
9. Opinion
10. Illustration

(Similarly, Easterbrook [East06] compared methods by rating them on a different set of ordinal scales, e.g., Environment: ‘[in the lab vs. in the wild]’ or Generalisation: ‘[statistical vs. analytical]’.)

Given that an expert’s opinion may be more insightful than a student’s poorly run field trial, ordered lists such as Dawson’s, Wieringa’s, or Easterbrook’s do not dictate the absolute value of research methods. Indeed, by Dawson et al.’s own admission, their “to some extent, subjective”
pyramid does not mean to imply that those at the top are superior to those at the bottom in every context. Rather, they indicate a preference order of research methods, which ultimately should be guided by the “consequential benefits if the method or tool [i.e., subject of evaluation] were judged beneficial and introduced, less the expected investment and introduction costs” [Kite96], as well as the consequences of reaching an incorrect judgement.

3.1.2 This Thesis’ Research Philosophy

The approach most suitable for this thesis appears to be an engineering approach to research, in the family of pragmatism and design science, since:

- The problem it examines is practical rather than theoretical, and it cannot be reproduced and experimented with in laboratory conditions (controlled, repeated, etc.);
- It entails actively developing artefacts (techniques and methods for specifying and analysing requirements for software systems), rather than passively and objectively proving theories;
- Different research methods may be required for the different phases of the research. E.g., surveys and interviews may be useful for problem identification, while action research and case studies may be useful for the implementation and evaluation of the proposed techniques;
- The pragmatic view that all knowledge is imperfect allows the thesis to contribute to the “chain of evidence” [RuHö08] about the studied phenomena. This is especially relevant in PhD projects, since time and money are limited, meaning that it would be difficult to answer research questions for the general population with statistical significance (i.e., for ‘all’ sociotechnical software engineering projects — if such a population could ever be identified).

However, this is not to say that the three other research philosophies have no part to play. Indeed, since this thesis’ topic concerns the development of sociotechnical systems, it:

- Aims at the improvement of software engineering practice to avoid undesirable events, which will undoubtedly involve challenging existing perceptions about that practice, and so the critical theory research philosophy is applicable;
- Involves different people who will have different opinions about the nature of problems or the usefulness of techniques, and so the interpretivist research philosophy is applicable;
- Is motivated by undesirable phenomena (e.g., that an implemented software requirement turns out not to be useful) that needs to be decomposed into potential causes in order to be studied. Hence, reductionist aspects of the positivist research philosophy are applicable.

3.2 Research Strategies

According to Hevner et al., — and as influenced by March and Smith [MaSm95] —, research in Information Systems can be characterised as either seeking to find “what is true” (behavioural science), or “what is effective” (design science) [HMPR04]. Behavioural science seeks to create, improve, and substantiate theories to explain human phenomena surrounding information systems’ analysis, design, implementation, management, and use. Design science, on the other hand, is a problem solving paradigm with emphasis on providing and evaluating solutions to those problems.

In the more specific context of Requirements Engineering research, Wieringa similarly proposes that research questions should be classified as either ‘design problems’ (i.e., creating effective artefacts) or ‘knowledge problems’ (i.e., pursuing justified truth) [WiHe08]. The fundamental implication of this is that requirements engineering research outputs should be evaluated either by comparison to stakeholder requirements, or by their correctness and integrity. Unfortunately for the field, the
majority of previous Requirements Engineering research has been ‘design’ (i.e., models, methods, and instantiations) rather than ‘design science’ [Maid11, WiHe06]. This is visibly apparent in RE papers where researchers (often) leave evaluation of their work in real life contexts ‘for future work’. (Perhaps due to the complexity of involving industry in academic research, the saleability of academic artefacts, a lack of immediate benefit to industry, legal and privacy concerns, and so on).

Rather than being mutually exclusive and orthogonal, design problems and knowledge problems are complementary and interconnected. Indeed, a loop exists between the development of theory and of artefacts, and vice versa [Wier09a]. Acknowledging the existence of this loop, Rossi and Sein add ‘better theory’ [RoSe03] to Hevner et al.’s list of design science outputs (later published in [HMPR04]). Hence, the outputs of design science research can be classified as the following:

1. **Constructs or concepts**: abstractions and representations that define the term used when describing an artefact;
2. **Models**: abstractions and representations that are used to describe and represent the relationship among concepts;
3. **Methods**: sets of steps and practices that are used to represent algorithms, processes or approaches on how to perform a certain task;
4. **Instantiations**: implemented or prototyped systems that are used to realise the artefact
5. **Better theory**: better understanding of relationships, concepts and processes as a result of the practice (e.g., through analysis and evaluation of outputs 1-4).

To conclude, this thesis primarily follows the design science philosophy (as aforementioned and justified in Section 3.1.1), which is inspired and rooted by the engineering cycle’s V process model [FoMo95], and is pragmatic and problem-oriented in nature. As such, the elicitation, creation, motivation, improvement, and evaluation of the constructs/concepts, models, methods, and instantiations (outputs 1-4) that are concerned with ‘the alignment of software requirements to their expected benefits [i.e., their sources of instrumental value]’ constitutes this thesis’ research strategy. A contribution of the fifth design science output (theory) is not explicitly claimed, since the intention is not to postulate a falsifiable theory based on the artefacts. However, that is not to say that there will be no such contribution whatsoever, since answers to numerous ‘Knowledge Problems’ (what is true?) on the subject are necessary for the effective design of artefacts [Wier09a].

### 3.2.1 Research Strategies for Advancing Requirements Engineering

Cheng and Atlee interpret ‘Research Strategy’ in Requirements Engineering research as ways of advancing the state of the art [ChAt07]. A list of such strategies, as synthesized from numerous Software Engineering and Requirements Engineering research guidelines, are enumerated in Table 3.5. The pertinence of each strategy to this thesis is evaluated and denoted in the rightmost column.
<table>
<thead>
<tr>
<th>Research Strategy</th>
<th>Definition</th>
<th>Pertinence to this Thesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Paradigm Shift</td>
<td>Dramatically change the way of thinking, resulting in a revolution in knowledge or technology.</td>
<td>Low</td>
</tr>
<tr>
<td>2. Leverage other disciplines</td>
<td>Leverage and recast principles, practices, processes, or techniques from another discipline.</td>
<td>High</td>
</tr>
<tr>
<td>3. Leverage new technology</td>
<td>Make advances by leveraging new tools or technology.</td>
<td>High</td>
</tr>
<tr>
<td>4. Evolutionary</td>
<td>Make progressive improvements to existing research solutions and techniques.</td>
<td>Medium</td>
</tr>
<tr>
<td>5. Domain-specific</td>
<td>Develop a new solution or technique that applies narrowly to a specific problem domain.</td>
<td>Low</td>
</tr>
<tr>
<td>6. Generalization</td>
<td>Generalize an existing solution or technique, so that it applies to a broader class of problems or data.</td>
<td>Low</td>
</tr>
<tr>
<td>7. Engineering</td>
<td>Develop processes or strategies that make it easier or cheaper to apply research solutions in practice.</td>
<td>High</td>
</tr>
<tr>
<td>8. Evaluation</td>
<td>Evaluate existing research solutions – with respect to specified metrics, real or realistic problems, current practices, or related research results.</td>
<td>Medium</td>
</tr>
</tbody>
</table>

The reasons why research strategies 2, 3, and 7 in Table 3.5 are considered to be of high pertinence to this thesis are as follows:

1. **Leverage other disciplines:** The notion of means and ends is certainly not specific to Software Engineering, and therefore it is expected that work from other disciplines would be highly relevant to treating and analysing ‘software requirements as means’.

2. **Leverage new technology:** It is expected that the process of eliciting, modelling, and reusing knowledge about means and ends within the Requirements Engineering context can be assisted by advances in software technology, especially in model driven engineering, simulation, visualisation, information retrieval, and recommendation systems.

3. **Engineering:** Comprehending the chain of means and ends spawned by a software requirement is a human activity, which could (and should) be assisted through practical improvements to processes, guidelines, and techniques.

### 3.2.2 Qualitative versus Quantitative, and Inductive versus Deductive

The final relevant interpretation of ‘Research Strategy’ is concerned with the choice between Qualitative and Inductive (mostly associated with interpretive research) versus Quantitative and Deductive (mostly associated with positivist research) research [EdTa99]. For brevity on this quite generic topic, the pragmatic research philosophy’s stance is referred to. Hence, if a research question calls for a quantitative study (e.g., if cognitive biases are likely to distort the validity of a survey question), and if the costs of quantitative research can be borne, then quantitative research should be favoured. An example of such a scenario might be determining programmer productivity: A qualitative approach might ask the practitioners themselves (in order to gauge the social perceptions of the phenomena), while a quantitative approach might more objectively examine the number and size of the practitioners’ contributions to the software’s source code version control system.

This thesis will follow a ‘Mixed methods’ research strategy (i.e., comprised of both Quantitative and Qualitative research). Indeed, some research questions may benefit from various sources of data to support their answer to ensure robust conclusions, especially where qualitative data is relied upon (i.e., conclusions should be ‘triangulated’ [RuHö08]). Hence, this thesis will fluctuate between
3.3 Research Design, Methods, and Process

This section details how data will be collected and analysed in order to answer the research questions. Unfortunately, the literature uses terminology such as research design, methods, instruments, processes, and tools interchangeably to refer to quite different concepts [Brym04]. According to Bryman, research ‘design’ should be used to refer to the abstract framework for data collection and analysis (e.g., Grounded Theory or Action Research), while research ‘method’ should be used to refer to techniques, instruments and tools for doing data collection and analysis (e.g., Structured Self-Administered Online Questionnaires or Telephone Interviews) [Brym04]. Finally, the research ‘process’ refers to the sequenced order in which the techniques will be performed. This thesis’ research design is discussed in Section 3.3.1, its research methods in Section 3.3.2, and its research process in Section 3.3.3.

3.3.1 Research Design

Numerous frameworks for research have been proposed by researchers, but those most notable (and those shortlisted in [Brym04, East06]) are: Action Research [Bask99], Case Study Research [Yin02], Ethnography [Myer11], Grounded Theory [GlSt67], and Experimental Research [Basi93a].

**Action Research** [Bask99] involves interactively participating in an “organisation change situation” while conducting research (where the importance of both activities is equally split [East06]).

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Avoids “armchair engineering” [Wier05, p.306] where solutions for practical problems that were never encountered by the researcher are proposed, to be later left un-validated and abandoned;</td>
<td>• Researchers can become distracted by the practical work, to the detriment of the research;</td>
</tr>
<tr>
<td>• Well suited for ‘how to’ research questions, since it requires the researcher’s intervention in a practical context (the researcher must ‘do’ as well as ‘see’);</td>
<td>• Costly due to time being spent on non-research tasks, and due to privacy, legal and ethical issues between the organisation(s) [East06];</td>
</tr>
<tr>
<td>• Provides high fidelity data due to detailed observations and researcher involvement;</td>
<td>• The ‘change situation’ may be affected merely by the researcher’s involvement (the ‘Hawthorne affect’ [Land58]), e.g., practitioners may increase adherence to best practice while being observed;</td>
</tr>
<tr>
<td>• Freedom to use any appropriate data collection methodology (the rigour required depends on the aims of the research and its stakeholders).</td>
<td>• The researcher’s social and political involvement with the practical work may distort the practitioner’s evaluations of the researcher’s designs or hypothesis (e.g., being positive to be kind).</td>
</tr>
</tbody>
</table>

**Case Study Research** [Yin02] involves in-depth examination of (discovery, testing, explaining, or comparing) a single instance or event (or ‘contemporary phenomena’), especially where boundaries between phenomena and context are not clear.
Table 3.7: Pros & Cons of the ‘Case Study’ Research Design

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Allows ‘smaller’ research contributions (i.e., with limited resources) to contribute to a ‘chain of evidence’ [RuHi08], providing additive conclusions, rather than having to provide wholly complete and self-sufficient conclusions;</td>
<td>• Some believe representativeness to be absolutely irrelevant in case studies [Mitc83]. Consequently, the ability to convince others of a case study’s objectivity, and acceptability is weakened [Flyv06];</td>
</tr>
<tr>
<td>• Just one case is sufficient for falsification of a theory, e.g., ‘all swans are white’ [Flyv06, p.11];</td>
<td>• Without replication, isolated or sparse case studies may provide inaccurate conclusions reached purely due to contextual factors or chance. E.g., a mathematically skilled workforce might find formal specification superior to Fagan Inspections for reducing software defects, but due only to the context rather than the approaches [Kitc96];</td>
</tr>
<tr>
<td>• Applicable where reductionism or experimentation is difficult, e.g., due to lack of control over context, variability of expected results, or limited resources;</td>
<td>• Generalising results to other contexts can be challenging, but it is necessary if it is to be used for support a theory, (as opposed to being a vehicle of description for a local phenomena [Phal95, p.41]);</td>
</tr>
<tr>
<td>• Well suited for ‘how’ &amp; ‘why’ questions due to consideration of context and depth of enquiry, i.e., for understanding the studied phenomena’s causes and effects [East06]; “More discoveries have arisen from intense observation than from statistics applied to large groups” [Flyv06, p.10];</td>
<td>• There is little agreement on what they are, what they constitute, and guidelines on them [Kitc96].</td>
</tr>
<tr>
<td>• Freedom to use qualitative or quantitative methods, and conclusions can support or generate a theory [Gerr06, p.33, HuMi02].</td>
<td></td>
</tr>
</tbody>
</table>

Ethnography [Myer11] involves interpretive (and strictly observational) study of phenomena through the meanings people assign to them, often requiring immersion in the studied social group.

Table 3.8: Pros & Cons of the ‘Ethnography’ Research Design

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Observations of the subject’s practice can be more revealing than their own perception of it, e.g., due to cognitive biases or politics (i.e., ‘external eyes’);</td>
<td>• Costly for the researcher due to extensive observations, especially since the organisation isn’t receiving any value (labour or valuable knowledge) during the study;</td>
</tr>
<tr>
<td>• Explores cultures in the state of practice;</td>
<td>• Generalisation can be challenging due to the high focus on context;</td>
</tr>
<tr>
<td>• Contrary to the previous point, observations can also be less accurate than reality (e.g., as ascertainable by work output analysis), due to reasons such as the Hawthorne effect [Land58].</td>
<td>• It may be frustrating to the researcher who may want to (or be expected to) provide input or feedback in real-time.</td>
</tr>
</tbody>
</table>

Grounded Theory [GlSt67] involves inductive development of theory or hypothesis from rigorously collected data through codification and categorisation to generalisable guidelines.

Table 3.9: Pros & Cons of the ‘Grounded Theory’ Research Design

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Since the theory follows data rather than researcher intuitions [SmHL95], the likelihood of generating a ‘correct’ theory is reasonably high compared to other research designs, at least bounded to the context of that data (i.e., not considering external validity).</td>
<td>• Data codification and categorisation can be very time consuming and subjective, especially where qualitative methods such as interviews are used;</td>
</tr>
<tr>
<td></td>
<td>• Data codification and categorisation can be very time consuming and subjective, especially where qualitative methods such as interviews are used;</td>
</tr>
</tbody>
</table>

Experimental Research [Basi93a] involves investigation of a falsifiable hypothesis through isolation of variables, adjustment of independent variables, and monitoring of dependent variables.
Table 3.10: Pros & Cons of the ‘Experimental’ Research Design

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Repetition of experiments ensures that conclusions are not reached due to underlying or unknown causes, or due purely to chance;</td>
<td>• Difficult to apply if the ‘right conditions’ [East06] cannot be simulated in lab conditions, e.g., if the context/environment cannot be isolated, and repeated in the same conditions as previous runs;</td>
</tr>
<tr>
<td>• Results come from a rigorous and well respected scientific method designed to isolate cause and effect. Hence, people are most likely to trust conclusions reached by experimental research.</td>
<td>• Deliberate isolation of a phenomenon from its context ignores that context can greatly affect phenomena in sociotechnical environments;</td>
</tr>
<tr>
<td></td>
<td>• Costly due to the need for repetition of experiments.</td>
</tr>
</tbody>
</table>

This thesis’ research includes neither theory building nor explicit falsifiable hypothesis (as in behavioural science or grounded theory). Furthermore, since the phenomenon exists within sociotechnical systems (both in humans developing software in Software Engineering, and humans using software in Information Systems), it cannot be easily simulated in laboratory conditions where variables can be isolated and context/environment can be controlled. For example, during repetition of the experiments, the same participants cannot ‘unlearn’ aspects of artefacts that may benefit competing artefacts. Furthermore, repetitious experiments with industry practitioners would be expensive, with little upfront benefit to them. (This often justifies the use of students acting as software engineers in experiments, but this tends not to be held in high regard [Wier05], since the context of the experiments is entirely different to where the evaluated artefacts would be used). Hence, while experiments would provide the most robust conclusions about the cost/benefit aspects of the artefacts, the resource limitations of the PhD project hinder their applicability.

Action research, case studies, and ethnography are the remaining viable research designs, and so a mixture is pursued in this PhD project. Of these, case study and action research seem the most applicable, since ethnography is not suited for the researcher’s involvement with the practical problem; Indeed, according to Baskerville, a newly invented technique or method cannot be studied without intervening to ‘inject’ it into the practitioner’s environment [Bask99]. Nevertheless, ethnography is viable for the early phases of the research project, in order to understand potential problems and their context in practice. Particular aspects of Action Research are most applicable for the formulation of this thesis’ artefacts, especially for its planned interaction with industry for practical and relevant problem identification, ‘how to’ research, and solution evaluation. However, Case Study research is most applicable for the exploratory aspects of the thesis: for understanding why the phenomenon occurs, and for describing the evaluations of the artefacts with rich context. Finally, this choice of research designs may be refuted by the reader, perhaps based on common misunderstandings of case study research, or more generally of ‘research by individual cases’ as opposed to by large scale samples and experimentation. In such cases, the reader is referred to Flyvbjerg’s extensive discussion of ‘Five Misunderstandings About Case-study research’ [Flyv06].

(Note that in practice, there is little difference between Case Study research and Action Research. In support of this, is that Runeson & Host synthesize four types of Case Studies in Software Engineering research: Exploratory, Explanatory, Improving, and Descriptive [RuHö08]; Their ‘Improving’ type (“where the researcher actively tries to improve the studied phenomenon”) appears to be almost identical to Action Research in its definition. Furthermore, Easterbook’s structured comparison of research designs also concludes that the only significant difference is that Action Research is slightly more data driven, as opposed to theory driven [East06].)
3.3.2 Research Methods (for Data Collection)

Literature Reviews are an essential phase of research that builds background knowledge on a subject, allows identification of knowledge gaps, motivates research design decisions, and provides ‘useful leads’ to maximise research effectiveness [KPBT09].

<table>
<thead>
<tr>
<th>Table 3.11: Pros &amp; Cons of the ‘Literature Review’ Research Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
</tr>
<tr>
<td>• Learning what others have done avoids ‘re-inventing the wheel’ (wasting research time);</td>
</tr>
<tr>
<td>• Learning what others have not done enables the researcher to efficiently contribute new knowledge;</td>
</tr>
<tr>
<td>• As a form of document (desk) research [Scot06], it does not heavily depend on the cooperation of others.</td>
</tr>
</tbody>
</table>

Self-Administered Surveys (Online Questionnaires) are a cost effective means of gathering beliefs, opinions, perceptions, characteristics and behaviours from a population [Umba04]. They are commonly performed as cross-sectional studies (performed at one point in time), rather than longitudinal studies (performed at many points in time) [Lavr08, p.171].

<table>
<thead>
<tr>
<th>Table 3.12: Pros &amp; Cons of the ‘Self-Administered Survey’ Research Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
</tr>
<tr>
<td>• Once the survey is created and distributed, the researcher’s involvement in the data collection and codification is minimal. Consequently, the sample size of the survey can be large at low cost;</td>
</tr>
<tr>
<td>• Surveys make use of existing experience; Data is recalled and made explicit, rather than being created solely for purpose of the research [Kitc96];</td>
</tr>
<tr>
<td>• Statistical analysis of the results is often straightforward due to the scope for question structuring and for closed (pre-set) responses.</td>
</tr>
<tr>
<td>• Many and wide-ranging self-reported variables (e.g., demographics, habits, etc.) can be compared at little cost (following a cross-sectional study design).</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Structured & Unstructured Interviews are a qualitatively rich means of capturing detailed responses from practitioners [JoCh06]. In an unstructured interview, the discussion can be guided by general themes of interest, or in a structured interview by pre-defined questions.

<table>
<thead>
<tr>
<th>Table 3.13: Pros &amp; Cons of the ‘Interviews’ Research Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
</tr>
<tr>
<td>• Questions can be clarified during the interview, hence reducing the risk of misinterpretation [Koth04, p.98] and therefore internal invalidity;</td>
</tr>
<tr>
<td>• Detailed opinions, stories, and examples are more likely to be elicited than with questionnaires, since two-way conversation is more thought provocative;</td>
</tr>
<tr>
<td>• More honest answers may be elicited than with non-face-to-face methods, since the interviewee may be ‘caught off guard’ when answering questions.</td>
</tr>
</tbody>
</table>

Proof of Concept is a method for collecting data to be used to validate a practical artefact resulting from research [HAILJ06], which in the software engineering research context is often a framework, process, system design, or algorithm manifested as an operational software program.
Table 3.14: Pros & Cons of the ‘Proof of Concept’ Research Method

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Without creating a working version of a design, unforeseen problems are likely to occur in real life. Hence a design’s robustness can be increased;</td>
<td>• Creating operational proofs of concepts consumes time and effort, often spent on technical implementation issues rather than conceptual issues;</td>
</tr>
<tr>
<td>• ‘Just enough’ functionality can be built to convince the audience that the design can be effective [HAHJ06], to perhaps be later built upon by others;</td>
<td>• The academic state of the art may be more advanced than the state of practice, and so the proof of concept may be of no more use than a vague design if the users cannot relate to it;</td>
</tr>
<tr>
<td>• A concept may be poorly understood by its evaluators until it is operationalised into a more physical manifestation or example, e.g., due to the ‘I’ll Know It When I See It’ effect [Boeh00]. Therefore, proofs of concept can increase the validity and integrity of the evaluation of the design.</td>
<td>• The difference between a proof of concept and a workable software tool acceptable for industry use is large. Usability, performance and security considerations (amongst others) may prohibit its use in an organisation, even for research purposes.</td>
</tr>
</tbody>
</table>

Expert Opinion is a data collection method typically used for evaluation [Brig91], whereby an expert is asked to render their professional opinion about the quality or cost-benefit ratio of an artefact such as a product or a procedure.

Table 3.15: Pros & Cons of the ‘Expert Opinion’ Research Method

<table>
<thead>
<tr>
<th>Pro</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Expert opinion is popular and affordable (relative to ‘objective-based evaluation’ [Brig91, p.228]), due to its minimal formality and structure;</td>
<td>• Evaluations are subject to an expert’s personal biases and traditionalism, and so can vary [WiFa83];</td>
</tr>
<tr>
<td>• Experts may have wisdom about the suitable criteria for determining quality, and so the validity of an expert evaluation may be better than ‘objective-based evaluation’ designed by the researcher or a smaller set of experts. In other words, experts can usually provide insights for decision makers that are absent from more objective methods [Brig91, p.228].</td>
<td>• Without explicit criteria (goals) for evaluating an artefact’s quality, an evaluation may merely capture the expert’s subjective ‘gut feeling’ about it;</td>
</tr>
<tr>
<td></td>
<td>• The quality of an artefact is often considered relative to expectations and alternative artefacts, which may be held or perceived inconsistently amongst experts;</td>
</tr>
<tr>
<td></td>
<td>• The robustness of an expert evaluation depends on the credibility of the expert, and the most credible experts will not always be available.</td>
</tr>
</tbody>
</table>

While the above five methods are the main methods to be used by this thesis, others exist (amongst 54 according to a recent count performed by Holz et al. [HAHJ06]) that would useful for particular research questions and contexts. For example, in order to answer research questions on current practice in software engineering, examination of physical artefacts such as code or documentation could be employed [SaKL97] to complement the more subjective surveys of practitioners. Indeed, in Chapter 6, requirements documents from real industry projects are statistically analysed to complement the more subjective findings elicited from surveys of industry practitioners.

3.3.3 Industrial Partners

Two large multi-national organisations were contacted to enquire about the possibility of this thesis’ researcher conducting the aforementioned ethnographic, action, and case study research. Given that the research problem exists in an industry context, this was considered essential to be able to conduct pertinent primary research. Consequently, the researcher was located in the offices of a large aeronautical engineering company (henceforth referred to as Case A) for one and a half years, and at a defence consulting company (Case B) for one year. The rationale for integrating with the day-to-day life of the organisations was that a more thorough understanding of the problems and their context would be afforded. (Retrospectively, interesting opinions and perceptions were learned from informal conversations and meetings acting as an ‘outsider’ to the organisation.)
Case A’s business unit produces and implements software to automate the design of civil aero-engine components, while Case B’s produces or implements information management software systems for military applications. (These case studies are described in more depth in Chapter 6.)

### 3.3.4 Research Process

Vaishnavi and Kuechler applied Takeda et al.’s analysis of reasoning during a generic design process [TVTY90] to design science [VaKu07], resulting in the framework for research shown in Table 3.16.

<table>
<thead>
<tr>
<th>Process Steps</th>
<th>Outputs</th>
<th>Knowledge Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Awareness of Problem</td>
<td>Proposal: Identification &amp; definition of the problem and its context.</td>
<td>Circumscription ↑</td>
</tr>
<tr>
<td>2. Suggestion</td>
<td>Tentative Design: Preliminary solution to the problem.</td>
<td></td>
</tr>
<tr>
<td>3. Development</td>
<td>Artefact: Solution developed with many iterations and feedback.</td>
<td>Operational &amp; Goal Knowledge ↑</td>
</tr>
<tr>
<td>5. Conclusion</td>
<td>Results: Conclusion and communication of research effort.</td>
<td></td>
</tr>
</tbody>
</table>

As depicted in Table 3.16’s research process, producing and evaluating the framework (in Chapters 5 & 6) increases awareness of the problem and its pertinent concepts, thereby influencing the contents of the literature review (Chapters 2 & 4) and vice versa. (I.e., cyclic induction & deduction is used).

### 3.4 Summary

This chapter has evaluated the relevance and applicability of research philosophies, designs, methods and strategies to this thesis. The philosophy most suitable for answering the research questions was judged to be the pragmatic philosophy in accordance with design science (Section 3.1.2), since the problem is practical rather than theoretical, and involves the creation of practical artefacts as well as knowledge. However, the subjective ‘human aspects’ of interpretivism and the reductionist aspects of positivism are unavoidable, given that the thesis’ topic is on sociotechnical software, and that the problem of optimising a software requirement’s value has many complex components.

This thesis’ research questions will principally be answered by the case study research design (3.3.1) due to its practical and non-replicate-able nature, and will involve the research methods of literature reviews, surveys, interviews, and proofs of concept (Section 3.3.2). Consequently, this thesis lies toward the interpretivist end of the positivist-interpretivist spectrum (Section 3.1.1). It uses ‘mixed methods’ to ‘triangulate’ its conclusions, and both inductive and deductive reasoning to evolve its research questions (Section 3.2.2). Finally, the most pertinent of Cheng & Atlee’s research strategies for advancing the field of requirements engineering were concluded to be ‘Leverage other disciplines’, ‘Leverage new technology’, and ‘Engineering’ (Section 3.2.1).

The next chapter aims to evaluate the state of the art approaches proposed in the literature that could be applied to treat this thesis’ problem, hence aiming to answer RQ3.
4. Review of Current Value-Based (Means to an End) RE Approaches

“If I have seen further, it is by standing on the shoulders of giants.” Isaac Newton
— “In computing, we mostly stand on each other’s feet.” Richard W. Hamming
This chapter reviews existing approaches that could be used to model the value generated by software requirements for their stakeholders. The first sections set the criteria for including (4.1) and comparing (4.2) the relevant approaches from the literature. The following sections individually summarise and evaluate the various approaches (4.3-4.8). The final section compares the approaches in a feature analysis table (4.9), ultimately concluding on the literature gap.
4.1 Inclusion Criteria

According to Kitchenham et al., literature reviews on software engineering approaches should have a well-defined inclusion criteria (i.e., reasons why a approach is reviewed or not) to ensure unbiased coverage [KiCh07, p.18]. As recommended [KiCh07, p.19], this review’s inclusion criteria was derived from the focus of the research questions in the Research Methodology chapter. Risk of selection bias (especially inherent in single-person literature reviews) has been mitigated by discussion with experts and advisors via supervision meetings, workshops, conferences, and peer-review.

To be included in the review, a paper:

1) Must be on an approach or a technique\(^{19}\) which could either:
   a) be used to explore, describe, or model the alignment of individual software requirements (as opposed to whole systems) to their expected benefits, i.e., where the requirement is treated as a means to some beneficial end;
   b) assist the means-end modelling process (1a).

2) Must be plausibly useful and usable in real world practice, i.e., the approach described has potential for practical application rather than for theoretical understanding.

3) Must be credible and understandable, i.e., the approach described must be expressive and intuitive to follow, as well as its published source being written in English.

The search strategy included:

- Searches in bibliographic databases (IEEE Xplore, ACM Digital Library, INSPEC, Compendex, ISI Web of Science, Elsevier, ScienceDirect, CEUR-WS, arXiv, SpringerLink, Wiley InterScience, and SCOPUS). This covers the vast majority of publications in Software Engineering and Information Systems research [KiCh07, p.25].

- Searches in Google Scholar for publications that either:
   - cite, or are cited by the key literature (i.e., ‘snowballing’ [JaWo12]);
   - are not in the above databases, e.g., for technical reports or other papers.

- Manual inspection of journal, conference, and workshop proceedings for papers related by theme rather than citations or terminology, especially within the proceedings of focused workshops such as REBNITA (Requirements Engineering for Business Need and IT Alignment), VALSOT (Value-Based Software Traceability), BUSITAL (Business/IT Alignment and Interoperability), iStar (at CAiSE), MoDRE (Model Driven Requirements Engineering), and RePriCo (Requirements Prioritisation and Communication).

Terminology usage within Requirements Engineering is inconsistent [JuMF08], especially within relatively new subfields such as Value Based Requirements Engineering [AuWo07] and Goal Oriented Requirements Engineering (GOR) [Lams01]. Indeed, experienced researchers have described literature reviews in RE to be “especially cumberson” [YGMM10, p.576] due to the field’s fragmentation and wealth of overlapping and complementary work [KaLo03, p.13]. As such, the search strings used for the review were a result of a learning and adjustment process. For example, work on ‘prioritisation’ requires benefit estimation and so pertains to our primary inclusion criteria, and work on ‘dependencies’ explores the benefit of a requirement through advantage provided to others.

---

\(^{19}\) A ‘technique’ is a practical method applied to some particular task. An ‘approach’ is a systematic arrangement (usually in steps) of ideas or actions intended to deal with a problem or situation [ZoCo05].
4.2 Comparison Criteria

A “common framework is necessary to make any sort of comparative evaluation” [Kite96], and so a set of comparison criteria is defined in Table 4.1. This criteria enables a type of qualitative evaluation known as Feature Analysis, which maps various alternative approaches to a common set of “properties, qualities, attributes, characteristics or features” [Kite96]. Since choosing comparison criteria is “essentially a subjective activity and depends on the experience, viewpoints, and interests of the individuals or groups concerned” [Kite96], the criteria were informed by (but not based on):

a. the background literature on the concepts defined in the conceptual framework (Chapter 2);

b. interviews with the thesis’ industrial partners (proposing criteria of practical interest);

c. other researchers’ proposed guidelines for approaches of a similar nature, namely:
   o Eckartz’ requirements for a software business case development approach [Ecka12, p.40];
   o Babar et al.’s requirements for an IT strategic alignment modelling approach [BaWG11];
   o Horkoff et al.’s comparison criteria for GORE model analysis methods [HoYu11].

<table>
<thead>
<tr>
<th>#</th>
<th>Short Name</th>
<th>Criteria – The approach should be able to…</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Means-End</td>
<td>Represent the relationship between an individual software requirement and its predicted benefits in terms of it (former) being a means to a desired end (latter).</td>
</tr>
<tr>
<td>A1</td>
<td>Benefit-Abstraction</td>
<td>Abstract a requirement’s benefit to a variable number of benefits (i.e., where an end becomes a means), in order to contextualise complex benefits.</td>
</tr>
<tr>
<td>A2</td>
<td>Disbenefits-and-Obstacles</td>
<td>Represent disbenefits and obstacles to achieving a benefit at various levels of a requirement’s benefit abstraction, in order to highlight risks to be avoided or mitigated.</td>
</tr>
<tr>
<td>A3</td>
<td>Requirement-Decomposition</td>
<td>Refine and decompose a requirement into atomic concerns that can be assigned and enforced by one agent in the system.</td>
</tr>
<tr>
<td>A4</td>
<td>Functional-Requirements</td>
<td>Provide guidance on representing behavioural (functional) requirements, i.e., those with Boolean satisfaction.</td>
</tr>
<tr>
<td>A5</td>
<td>Non-Functional Requirements</td>
<td>Provide guidance on representing quality (non-functional) requirements, i.e., those with various possible prescribed target degrees of satisfaction.</td>
</tr>
<tr>
<td>A6</td>
<td>Benefits-As-Variable-for-Improvement</td>
<td>Provide guidance on representing expected benefits in terms of a numeric variable (with an objective interpretation, i.e., rooted in the application domain) to be influenced in a direction of acceptability on its measurement scale.</td>
</tr>
<tr>
<td>A7</td>
<td>Benefits-As-Magnitude-of-Improvement</td>
<td>Provide guidance on representing benefits in terms of unambiguous, measurable, time-specific, benchmarked, and stakeholder-owned objectives (as improvement to its variable in the application domain).</td>
</tr>
<tr>
<td>A8</td>
<td>Guide-Benefits-Elicitation</td>
<td>Provide guidance on eliciting the benefits of implementing a requirement, e.g., lists of (personal/consumer/business) values, or re-use from old projects.</td>
</tr>
<tr>
<td>B</td>
<td>Contribution-Extent-Score</td>
<td>Represent knowledge about the degree to which a means contributes to an end with either partial, full, or exceeding sufficiency and/or necessity.</td>
</tr>
<tr>
<td>B1</td>
<td>Contribution-Polarity</td>
<td>Represent whether a software capability is believed to influence an expected benefit (or disbenefit) positively or negatively, and ideally guide representation of bipolarity.</td>
</tr>
<tr>
<td>B2</td>
<td>Contribution-Refutable</td>
<td>Represent contribution knowledge falsifiably and refutably, e.g., in terms of estimated impact on an objective numeric scale rooted in the application domain.</td>
</tr>
<tr>
<td>B3</td>
<td>Contribution-Dose-Response</td>
<td>Represent knowledge about the effect (the response) of modifying the means (e.g., the requirement’s target level of satisfaction—the dose) on an end.</td>
</tr>
<tr>
<td>B4</td>
<td>Contribution-Non-Linear</td>
<td>Represent non-linear contribution in dose-response relationships, e.g., to show the scope for benefit maximisation and/or diminishing returns.</td>
</tr>
<tr>
<td>B5</td>
<td>Contribution-Uncertainty-By-Unknowns</td>
<td>Represent lack of knowledge (epistemic uncertainty) in the predicted contribution, i.e., the stakeholder’s degree of belief that implementing the requirement to some degree (the means) will lead to some degree of benefit (end).</td>
</tr>
</tbody>
</table>
To improve the structure of the literature review, the reviewed approaches have been classified under the broad taxonomy\(^{20}\) of:

- **Popular Requirements Specification Approaches** (Section 4.3) for eliciting and describing requirements in a typical software project, representing the un-specialised (inadvertent) exploration of a requirement’s value, e.g., for prioritisation;
- **Traceability Approaches** (Section 4.4) primarily co-relate software engineering artefacts, including those as means (e.g., software requirements) to ends (e.g., stakeholder requirements);
- **GORE Approaches** (Section 4.5) build on Goal Oriented Requirements Engineering languages and techniques for the exploring and aligning system goals and stakeholder goals.
- **IT Strategic Alignment Approaches** (Section 4.6) aim to achieve ‘line of sight’ from operational tasks (e.g., using software) to the creation of competitive advantage for an organisation (e.g., planned or emergent strategic goals [Mint00]);
- **Benefits Realisation Approaches** (Section 4.6.3) are generic to exploring and managing the realisation of benefit for a business investment or intervention;
- **(Software) Systems Engineering Approaches** (Section 4.8) that cannot be categorised under the previous categories, but employ modelling techniques to describe the pertinence and/or predict the behaviour of interventions in systems (software-based or otherwise).

The subsequent sections discuss the individual approaches that meet the inclusion criteria from Section 4.1, evaluating each by their applicability and pertinence to this thesis’ problem using the

| B6 | Contribution-Uncertainty-By-Variation | Represent irreducible variation (aleatoric uncertainty) in the predicted contribution, i.e., both the extent and the reasons for any variation in a stakeholder’s estimate of the benefit caused by implementing the requirement. |
| B7 | Contribution-Propagation | Propagate contribution through a chain of means and ends to answer ‘what-if?’ questions on the effects of requirementA on benefitZ through B,C,D,... |
| C | Agents | Represent and classify (e.g., ‘person X plays the role of Y’) ‘agents’ [DaLF93] (stakeholders, software components, etc.) related to a requirement or expected benefit. |
| C1 | Guide-Agent-Elicitation | Provide guidance on eliciting the stakeholders who could gain benefits or costs from a requirement, or who could contribute to the requirement’s satisfaction. |
| C2 | Agent-Responsibility | Represent that agents are causally and/or consequentially responsible for the satisfaction of a means or an end. |
| C3 | Agent-Desire | Represent that agents desire the achievement of an end (i.e., to answer ‘who benefits from the requirement’s implementation?’) |
| D | Explicit-Assumptions | Represent any assumptions that provide the necessary criteria for a means having the stated contribution toward an end. |
| D1 | Assumption-Agreement | Guide the elicitation of stakeholders’ agreement (i.e., variation in belief) on the correctness of any assumptions made, including means-end contribution scores. |
| E | Utility-Satisfaction | Represent the utility and disutility of satisfying or not satisfying an end, normalised (weighted) for comparison between the whole set of ends. |
| F | Tool-Support | Support the implementation & ease the complexity of the approach with a software tool. |
| F1 | Tool-What-If Answering | Support the answering of ‘To what extent are directly and indirectly related benefits affected if the requirement is adjusted?’ questions using a software tool. |
| F2 | Tool-Reuse | Support the elicitation of a requirement’s benefits and disbenefits (outlined in criteria A & B) via automated retrieval & re-use of relationships within previous project models. |

---

\(^{20}\) No claims are made about the mutually exclusivity of the taxonomy (e.g., a GORE approach has some aspects of Traceability). It is merely a ‘good enough’ classification to structure this review.
comparison criteria from Section 4.2. Ultimately, Section 4.9 presents the feature analysis which summarises all of the discussed approaches.

4.3 Popular Requirements Specification Approaches

Before reviewing specialised approaches for exploring the value of a requirement, it is useful to understand the extent to which it occurs in the mainstream RE approaches. The approaches selected to represent this category were chosen based on:

- surveys of RE technique usage, e.g., which find that “many popular techniques in the research community ... are almost unknown in industry” [PKBA05, p.428];
- surveys of industry RE tool features, e.g., which find that “requirements modelling and management are the most badly supported categories of features” ... “whereas natural language statements ... are usually well covered by the RE tools” [CNFT12, p.13];
- surveys of the contents of university RE courses [Lock16, p.11], e.g., which find that the majority of surveyed courses do not teach modelling techniques such as SysML or GORE;
- their endorsement in general software engineering textbooks [Somm11a, p.97,230].

4.3.1 Volere Requirements Process & Natural Language Template

Robertson and Robertson’s popular Volere process and template for specifying requirements [RoRo12b] is not especially focused on the exploration of value, but it embodies a good practice requirements engineering process, and so is representative of requirement-value exploration in a typical project. Reassuringly, Volere’s advocated requirement template fields and its advocated processes for populating them are in agreement with various international RE standards, textbooks, and guidelines, e.g., [GoKB08, HuJD11, Icee11b, Icee11a, MaTh13b].

Firstly, Volere proposes that the overall goals of the software project should be described such that each goal “is firmly established, is reasonable, and is measured”, and that they “quantify the advantage gained by the business through doing the project” [RoRo12b]. Crucially, the “advantage gained by the business” is described for the project, rather than for discrete software capabilities (requirements). Additionally, Volere proposes a ‘one sentence’ natural language representation of goals to describe the project’s “Purpose, Advantage, Measurement (PAM)” [RoRo12b, p.8]. However, due to their unstructured natural language representations, goal specifications may be ambiguous, where, for example, a goal’s timeframe, scale, or benchmark attributes [Gilb05a] may be missing or unverifiable. Furthermore, neither a visualisation nor a structure between goals and requirements is proposed, which makes representing and comprehending relationships such as dependencies, contributions, decompositions, or generalisations [DaLF93] difficult. As such, understanding goals in their context is not aided: e.g., knowing ‘what will happen as a result of the advantage (i.e., why is it instrumentally valuable)?’, or more importantly, ‘which requirements contribute to it?’

Secondly, Robertson & Robertson propose that individual requirements should be comprised of fields, as defined by the ‘Volere requirements shell’ [RoRo12b]. Especially relevant to this thesis are: rationale, fit criterion, priority, customer satisfaction, customer dissatisfaction, and supporting materials.

Volere’s rationale field is intended to provide a justification for implementing the requirement, which along with the fit criterion field helps to “expose false constraints (solutions masquerading as constraints)” [RoRo12b, p.17]. Hence, the rationale field might be considered a description of the desirable end that the requirement is a means to, and hence a description of how the requirement
Popular Requirements Specification Approaches

is instrumentally valuable for that end. However, there is an important difference between a justification among alternative choices (i.e., substitutable means to an end), and a description of the benefit of the requirement’s outcome (i.e., the end). For example, a description of the value created for a stakeholder by having a Graphical User Interface relative to the ‘do nothing’ alternative is different to the value it creates as a substitutable means relative to alternative user interface implementations. (This is later elaborated by the distinction between i*’s goals and tasks in Section 4.5.1.) Furthermore, Volere’s rationale field tends to procure a shallow description (e.g., all given examples are one sentence [RoRo12b, p.17]), whereas instrumental value is created by software through a multi layered ‘web’ [RoGL08] of causation. Many requirements ultimately contribute to a smaller number of benefits through different chains of cause and effect, just as at a higher level of abstraction, “it is not unusual for more than one program to target the same set of benefits” [Thor99, p.120]. For example, it is unclear whether the rationale field is to describe benefits to a manufacturing process, the resulting benefits to the order fulfilment process, to customer satisfaction, to the customer re-order rate,... or all of the above. To express all of this using natural language within a requirement’s rationale field would cause repetition within the requirements document, causing redundancy, which is considered a requirements quality defect [Ieee98]. Conversely, not doing so would cause “failure to go deep enough into the problem context” [Jack00].

Volere’s fit criterion field should be a measurable, testable, unambiguous, and objective benchmark (or ‘acceptance test’) to determine if the requirement has been satisfied [RoRo12b, p.47]. Without it, claims about the benefit (effect) of implementing a requirement (cause) would be untestable. However, satisfaction of a fit criterion is considered to be Boolean (e.g., was average response time $\leq 30\text{ms}$ : true or false?), and so it would appear as though the requirement’s value proposition is ‘all or nothing’. Conversely, numerous RE guidelines stipulate that multiple levels of satisfaction should be set [Gilb05a, Maid06, PAPG09], e.g., for the current level (benchmark), the worst case, and the best case. Clearly, the benefit generated by the satisfaction of a requirement depends on which of these levels a requirement’s ‘satisfaction’ refers to, which is ignored by Volere. Furthermore, setting fit criteria that are precise and realistically achievable can be troublesome in practice. Juristo et al.’s survey on RE practice [JuMS02] concluded that there are two main obstacles to quantifying Non-Functional Requirements (NFRs). Firstly, unrealistically specified (unnecessarily demanding) requirements can lead to increased costs and delays (a common cause of cost increases [CIBa10]), and secondly, determining what is realistic with current technology to set realistically achievable targets (the ‘R’ in ‘SMART’ [CrLy88]) requires time consuming research. Volere provides no guidance on dealing with uncertain or evolving fit criterion, or ‘deidealising goals’ [Bana10].

Volere’s priority field is intended to provide the optimal order of implementation for the requirements (as visualised by Figure 4.1), and so one would expect them to embody judgement about the value a requirement would provide. However, priority is (or should be) a function of numerous variables, including urgency, benefit, cost, risk, volatility, knowledge-gain, etc. [HeBa06].

![Requirements Prioritisation Diagram](adapted from [HMSY12])

Figure 4.1: The Requirements Prioritisation Problem (adapted from [HMSY12])
Numerous experience reports note that Volere’s advocated priority scores of ordinal numerical scales (e.g., 1-5) or MoSCoW (Must do, Should do, Could do, or Won’t have\textsuperscript{21}) \cite{Iiba09} fail to capture the optimal priority of requirements (and hence neither an indication of their value) in projects with a large number of requirements (above 300) \cite{KBPP12}. Even in smaller projects, the distribution of priority ratings tends to be unhelpfully skewed toward ‘Must’, since stakeholders tend to be enthusiastic about their own requirements, feeling as though “specifying requirements is like filling up a shopping cart … at no cost” \cite{Eele05}, or they have learned from past experience that anything other than ‘Must’ tends to be scrapped \cite{Mavi14}. Hence, at the very least, if qualitative priority levels are used, they should be defined relatively rather than absolutely, i.e., ‘higher than’ rather than ‘high’ \cite[p.108]{Lams09a}. Regardless, requesting single priority ratings from stakeholders assumes that they are capable of determining the intrinsic value of each requirement, thereby implicitly comprehending, summarising, and weighting each chain of instrumental value possibly generated by the requirement into a normalised (and so cross-comparable) priority rating. This is a risky assumption, since stakeholders do not often understand the implications of benefit identification \cite[p.61]{SaHK08}.

Determining the relative value of a requirement for their prioritisation is also complicated by dependencies on other requirements. For example, the requirement ‘cut to clipboard’ might be rated as ‘Must’, but only if ‘paste from clipboard’ were also implemented. Grouping requirements into hierarchies “that take care of the important dependencies” \cite{HeDa13} was first proposed by Karlsson et al.\textquotesingle s prioritisation approach in 1997 \cite{KaOR97} — and grouping requirements with trees or graphs \cite[p.108]{Lams09a} is a now common ’best practice’ (as in the Core metamodel \cite{GoKB08}, SysML \cite{SoVr08}, GRL \cite{AGHM10}, KAOS \cite{DaLF93}, CTT \cite{PaMM97}), yet it does not exist in Volere. In addition to dependency, substitutability also affects the relative value of requirements. For example, the priority of ‘cut to clipboard’ might be reducible if ‘copy to clipboard’ were to be implemented. Hence, single subjective and non-relative ratings of priority, satisfaction, or value, are oversimplifications of a highly combinatorial solution space, since a rating is implicitly made in the context of (or ‘anchored’ \cite{TvKa73} by) a “reference system”, which may be imagined by the stakeholder as the status quo system, or as the “perfect system where all requirements are assumed to be implemented” \cite{HeDa13}. (A ‘reference system’ is defined as the stakeholder\textquotesingle s mental model of the whole set of requirements, representing one of many possible variations of the system-to-be \cite{HeDa13}. E.g., the dissatisfaction rating for a ‘cut’ feature would be different, depending on whether ‘copy’ is in the imagined reference system. As such, it might incorrectly appear as though stakeholders have different preferences about the requirements.)

The customer satisfaction and customer dissatisfaction ordinal scale (e.g., 1-5) rating fields are similar to, and were perhaps inspired by Kano\textquotesingle s model of satisfaction \cite{KSTT84}. A stakeholder can find positive value in having a requirement implemented (satisfaction), but they also find negative value in not having the requirement implemented (dissatisfaction). As Figure 4.2 shows, product features (implemented requirements) can be represented by one of the three curves, based on their relationship between the degree of implementation, and the degree of satisfaction provided. For example, an often taken-for-granted ‘save work’ software feature would likely be on the ‘Basic needs’ curve in Figure 4.2 due to its obviousness or implied presence, and so the requirement’s Volere dissatisfaction rating would be higher (e.g., 5) than its satisfaction rating (e.g., 2).

\textsuperscript{21} ‘Won\textquotesingle t have [until the next software release]’ is the lowest rank in MoSCoW \cite{Ash07, Iiba09, LGNR11, p.112}, but it is sometimes replaced with or ‘Would do’ or ‘Would be nice’. Other priority enumerations such as the IEEE Std 830’s \{Essential, Conditional, Optional\} \cite{Ieee98} are also popular.
Regardless of the above, both ‘basic need’ and ‘delighter’ requirements may actually provide equal amounts of benefit [HeDa13], meaning that the penalty of not implementing a requirement may objectively amount to a subtraction of the possible benefit in implementing it. Put simply, Kano’s model shows perceived difference, rather than actual (i.e., objective) difference in value realisation. For example, consider Cohn’s example of a legal requirement that would “bring no value to the company” and whose “only downside to not implementing the feature was that the CEO might spend some time in prison”, and so would have a “relative benefit [satisfaction score] of 1, but a relative penalty [dissatisfaction score] of 9” [Cohn05, p.116]. However, probabilistic counterfactual causal propositions [PaWe01] (as discussed in Chapter 2) could objectively describe such estimates with one number that answers both ‘what if?’ and ‘what if not?’. For example: ‘the probability of the CEO spending time in prison for violation of that particular law would be reduced by 0.799 (e.g., from 0.8 to 0.001) by the software feature’ (defined relative to what the probability “would otherwise have been” [Rich97]).

Since Volere’s (dis)satisfaction ordinal ratings are subjective assessments (i.e., they have “no physical interpretation in the application domain” [LeLa04]), the integrity of conclusions made from them may be affected by cognitive biases such as confirmation bias [Nick98] or the availability heuristic [TvKa73], political influences [MiMa12], interpersonal utility differences [Binn07], pre-conceived opinions un-adjusted for the new software context, or hidden assumptions based on a lack of or incorrect knowledge [JoHo06]. Such RE faults would be more easily correctable if estimates of the requirement’s value were explained objectively and transparently such that they are refutable by stakeholders or experts [Jack95a].

Volere’s supporting materials field enables traceability between requirements, use cases, tests, and other referable information that relates to the requirement. However, there is no semantic meaning in the traceability links specified within the Volere requirements shell, and so there is no way to distinguish, for example, that a requirement contributes to or conflicts with a related entity such as a benefit. In addition, textual representation of traceability is not as comprehensible as matrix or graph based visualisations [LiMa12]. Furthermore, little guidance is provided on what should and should not be related, and when to do so, which can lead to wasteful process compliance oriented traceability (i.e., for the sake of ‘best-practice box ticking’) [AsFT07], or just none at all.
Finally, advantages in communicating and analysing requirements using machine-processable models rather than structured [MWHN09] or unstructured natural language are well documented [BRLD94, Lams09a, p.127], e.g., for analysis, disambiguation, and re-usability. Nevertheless, the majority of industry practice tends to favour natural language [MaLa08, Petr13, PyOl13] specification, such as using Volere, due to its ease of comprehension [MoHM10].

Table 4.2: Feature Comparison Table (Key: ✔ Explicitly, ✅ Somewhat, ✘ Not) Supported

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Summary of Approach #1: Volere Requirements Process & Natural Language Template

Summary: 'Fit Criteria' provide an unambiguous test to decide if a requirement is satisfied. 'Customer (Dis)satisfaction' and 'Priority' ordinal ratings provide a comparison mechanism to judge the benefit of the requirement's (dis)satisfaction relative to other requirements. 'Purpose, Advantage, Measurement' project goals explain the benefit of the whole system.

Pros & Cons: The Volere Process has a relatively low cost of implementation (e.g., subjective rather than objective data is required for the satisfaction & priority ratings), but does not explain how a requirement is instrumentally valuable.

4.3.2 Agile Requirements (User Stories & Planning Poker)

The other mainstream alternative to Volere for describing requirements is Agile’s ‘User Story’ approach. Three contextual points must be raised before discussion can proceed:

- ‘Agile’ and ‘traditional’ techniques are mostly not incompatible nor conceptually distinct [PLMC13, p.23]. Indeed, the former are considered by some to be a ‘repackaging’ [MeTR05].
- Agile is not frequently used for large, ‘complex’ [Somm11a, p.84] or ‘mission-critical’ [Lams09b, p.54] system development, e.g., due to costs of frequent communication between end-users & developers [CaRa08], or a need for rigorous verification prior to coding (see [Lams09a, p.54] & [Buck10] for more). Less pragmatic reasons include cultural or legal constraints [PLMC13, p.73].
- The Agile requirements approach discussed here is Cohn’s [Coln05], which is regarded among the Agile community as one of the most comprehensive. However, less rigorous requirements approaches (or even none at all) are often followed by practitioners under the ‘Agile’ name, as is illustrated by the popular Dilbert cartoon [Adam07a] in Figure 4.3.
In Agile RE, a ‘product backlog’ containing ‘user stories’ is the equivalent of a traditional requirements document [CaRa08, PLMC13, p.29] containing lists of functional requirements defined from a user’s perspective [Cohn05, p.259]. Non-functional concerns are included among other “conditions of satisfaction” [Cohn05, p.125] attached to user stories, similar to the way in which traditional ‘use cases’ can be related to non-functional requirements [DKKP03].

Cohn advises that user stories should follow the syntax of “As a <user type>, I want to <goal> so that <reason>.” [Cohn05, p.259], where the ‘so that’ clause is intended to express the ‘business benefit’ [Cohn05, p.25] of the functionality entailed by the goal. However, the ‘so that’ clause is considered optional [Cohn05, p.260], and naturally tends towards descriptions of benefit in terms of low-level operational process goals rather than strategic goals or business improvements. Indeed, Cohn’s own set of example user stories illustrates its process orientation, where the ‘reason’ field of the first story becomes the ‘goal’ field of the next story [Cohn08]. As such, the previous criticism of the shallowness of Volere’s rationale field is applicable to the ‘reason’ field of a user story.

In Agile, the ‘business value’ of functionality is estimated for the purpose of prioritising user stories (in addition to estimating development cost, knowledge generated about the product and the project, and risks of schedule delay, cost overrun, or non-working functionality [Cohn05, p.80]). To reduce complexity (of size and dependencies), sets of related user stories are grouped into ‘themes’ (e.g., “import and export of data” [Cohn05, p.79]), and the business value of themes is assessed rather than for individual user stories [Cohn05, p.53]. I.e., software ‘features’ [CIHS08] are prioritised.

Cohn suggests two approaches for estimating the value of a theme: stakeholder desirability [Cohn05, p.81] and financial value [Cohn05, p.91]:

- Cohn’s stakeholder desirability approach is similar to Volere’s customer satisfaction rating approach (previously discussed in 4.3.1) since it is based on Kano’s model of satisfaction, but using the terminology of Wiegers’ prioritisation approach [Wieg03].
- Cohn’s financial value approach calculates time-adjusted financial measures such as Net Present Value and Discounted Payback Period [Cohn05, p.91] to assess the profitability of a theme to the business. However, estimating a single figure for the amount of money that will be made or saved by having new functionality is a complex and difficult task, since it depends on a myriad of assumptions and uncertainties — none of which are made explicit. Perhaps more critical is that a focus on financial benefit is ‘unbalanced’ since it ignores vital non-financial benefits [KaNo96b]. Finally, as with Volere’s customer (dis)satisfaction fields, the mechanism by which the requirement would create the claimed value is not made explicit. Consequently, the claims are not easily refutable and so are not a robust form of argument [Alex08] for the requirement being valuable.

In practice, Agile developers often spent more time estimating the cost (implementation effort) of functionality (its ‘size’ in ordinal ‘story points’) than its benefit, riskily assuming that user stories
are intrinsically valuable because they are user oriented. (As is van Cauwenbergh’s argument [Vanc09], and perhaps the reason why van Lamsweerde warns against using Agile techniques when the customer and end user are not the same person [Lams09a, p.54].) Indeed, the unqualified term ‘estimating’ is defined in Cohn’s book as “‘How big is this?’”[Cohn05, p.iii], and is similarly used throughout Agile literature to refer only to implementation effort. Even Cohn’s example of user story prioritisation has an ‘estimate’ only in terms of story points [Cohn05, p.138].

The final noteworthy but slightly tangential point is that Cohn favours the ‘planning poker’ approach for estimating unknown numbers such as story points [Cohn05, p.56]. Based on Wideband Delphi approach [Welt72], planning poker reduces the chance that one person’s estimate will be affected by another’s, and hence both ‘the wisdom of the crowd’ [Suro05] is improved and the ‘anchoring’ cognitive bias [TvKa73] is avoided. Additionally, planning poker makes estimation easier by restricting the set of possible estimates to the Fibonacci sequence [Cohn05, p.52], which is chosen because gaps in the sequence increase as the numbers increase. This is in recognition of the fact that humans are good at estimating things that fall within one order of magnitude [Saat96] (and so any exponential scale would suffice), and it also reduces cognitive load by providing the estimator with pre-defined options [Krug05]. Trivial debates about the difference between, e.g., 5 and 6 story points, are thus avoided. Furthermore and finally, by providing a quickly growing set of possible estimation values, the estimator can easily produce an interval estimate by eliminating the upper or lower values which seem infeasible to them [Hubb10]. Indeed, by exploring options that the estimate cannot plausibly be, a surprisingly informative estimate can be gained from people who claim at first to be unable (unknowledgeable) to provide an estimate [OhOa04, p.245]. Overall, planning poker (as one of Agile’s estimation techniques) could be useful in eliciting estimates for a probabilistic causal proposition to describe the size of the benefits of a requirement (e.g., the median system response time given a certain architectural option) under high uncertainty or time pressure.

Table 4.3: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Summary of Approach #2: Agile Requirements (User Stories & Planning Poker)

**Summary:** User stories are similar to Volere’s functional requirements (with rationale), but are user-oriented. Guidance on eliciting a financial value (£) for each story, and a prioritisation process to minimise estimator bias is provided.

**Pros & Cons:** Requirements are user-action oriented, so are less likely to create redundant functionality. However, user interaction does not entail value generation, since users are not always the intended beneficiaries. Single point estimates of financial value do not explain why or how value is generated, and favour description of tangible high-certainty benefits.
4.4 Traceability Approaches

Requirements traceability is likely to be at the core of any approach relevant to this thesis, since examining the instrumental value of a requirement calls for relating requirements to other entities that the requirement derives its value from.

4.4.1 MODAF SV-5 (Requirements Traceability Matrix)

Most authoritative requirements engineering guidelines and methodologies advocate the exposition of requirement relationships [Ieee11b, KoSo98, Kovi98, RoRo12b, SaRo09, SoSa97a], including between requirements, the sources of requirements (e.g., stakeholder meetings, documented business goals, policies, rules, laws, etc. [Ieee11a, p.111]), and also the implementation of requirements (e.g., software design and code) [GoFi94]. The semantics and data structures of the possible relationships between requirements are numerous and varied, ranging from tree structures (one parent, many children) describing functional decomposition according to reductionism [SBJA98], to graph structures (many parents, many children) describing cross-cutting concerns such as aspects [ZhJY05], dependencies [Yu95], conflicts [LaLe00], wishes [DaLF93], correlations [AGHM10], argumentation beliefs [Alex08], domain knowledge and assumptions [AtKM04], and more.

Regardless of the variety of requirement-relationship structures and semantics explored in the literature, the most popular approach in practice is a simple requirements traceability matrix [Ieee11a] (formally: an undirected graph adjacency matrix [Bued00, p.113]), most often with loosely defined ‘relates to’ link semantics. Traceability matrices are popular for ‘coverage analysis’ [HuJD11, p.15] to ensure that the system (including software) requirements that represent solutions [Ieee11b, p.27] will fulfil the needs defined by the stakeholder requirements that represent problems [Ieee11b, p.20]. A notable example of a standardised implementation of such a traceability matrix is the MODAF’s (Ministry of Defence Architectural Framework) System View 5 (SV-5) [Mini05]. SV-5 visually “acts as the ‘glue’” [Mini05] between system requirements and the stakeholder requirements that “are met by” [Mini05, p.18] them, and so it could be argued that SV-5 explains how system requirements create value by ‘meeting’ stakeholder requirements.

Traceability matrices provide a quick overview of all artefacts (requirements) and their links, and are easily understood by non-technical stakeholders [LiMa12]. They also show omission (i.e., lack of traceability visualised through empty cells) as well as commission, and so in SV-5’s case, they guide the elicitation of links from solutions to problems, and so to benefits. However, pairwise comparison of each system requirement against each stakeholder requirement scales poorly, and so the matrices can be time consuming to populate. Furthermore, the information in each trace in SV-5 (and most other traceability matrices) is Boolean, and so either the system requirement is or is not ‘met by’ the stakeholder requirement. However, in reality, individual system requirements often only partially ‘meet’ stakeholder requirements [LeLa04]. This causes difficulty in constructing and interpreting a matrix, since it is unclear how much of a contribution from a system requirement is required for the stakeholder requirement to be ‘met by’ it. As such, the matrix would be either sparsely or heavily populated, depending on the practitioner’s interpretation of ‘met’. (E.g., a ‘print’ functional requirement might be described to ‘meet’ a stakeholder requirement for ‘communication security’ due to its provision of offline, i.e., secure, communication of digital files. However, it is neither necessary nor wholly sufficient for ‘system security’, and so the link might be misunderstood.)

Furthermore, the description of benefit generated, as well as conclusions from later ‘impact analysis’ afforded by SV-5 — ‘what if the requirement is changed?’ [HuJD11, p.15], are likely to be ambiguous.
Overall, understanding how a system requirement contributes to a stakeholder requirement may be difficult without contextual information such as assumptions and argumentation (as is modelled by graph-based traceability approaches such as Hull et al.’s ‘Rich Traceability’ [HuJD11]). Finally, compared to graph-based traceability comprised of nodes and edges (or ‘boxes and arrows’), comprehending relationships between more than two entities is challenging using the matrix format.

Table 4.4: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>

Summary of Approach #3: MODAF SV-5 (Requirements Traceability Matrix)

Summary: Explains the benefit of a software requirement (solution) in terms of the stakeholder requirement(s) (problems) that it satisfies, via Boolean relationships recorded in intersecting cells of a 2d traceability matrix.

Pros & Cons: Traceability matrices shows omission (as well as commission) of links, but this causes poor saleability. A 2d matrix relates requirements to one other set of requirements, and so cannot show chains of instrumental value.

4.4.2 Software House of Quality (HoQ)

Conceptually similar to the MODAF SV-5 traceability matrix is Herzwurm et al.’s ‘Software-HoQ’ (House of Quality) [HeSP03]. As Figure 4.4 exemplifies the HoQ translating the ‘voice of the customer’ to the ‘voice of the engineer’ [HeSP03, fig.5], or in other words, ‘user requirements’ into ‘technical requirements’ [HeSP03, fig.2], or ‘needs’ (rows) into ‘solutions’ (columns) [HeSP03, p.1].

Figure 4.4: Example Software House of Quality for an e-mail client, adapted from [HeMa11]

The Software-HoQ is a customisation of the HoQ from Akao’s Quality Function Deployment (QFD) [Akao90], which has been successful in creating competitive advantage for traditional (i.e., tangible) product designers [HeSP03, p.1]. However, Herzwurm et al. claim that QFD’s focus in software engineering needs to be on earlier development phases, such as “setting prioritised development goals based on the most important customer requirements”[HeSP03, p.2], since, for example, the software
‘production’ phase is essentially duplication. They further motivate their ‘Software HoQ’ by observing that “software is valued not for what it is, but for what it does” [HeSP03], and so software functionalities and non-functionalities need to replace the HoQ’s physical solution characteristics.

Instead of MODAF SV-5’s recording of Boolean relationships in cells, the HoQ records a number between 0 (‘not expected’) and 9 (‘very likely’), corresponding to “the degree of correlation between the satisfaction of a customer requirement by a product function” [HeSP03]. However, while these correlation scores borrow credibility in name from objective statistical approaches (e.g., Pearson’s correlation coefficient [Aldr95]), the HoQ’s correlation scores are subjectively assigned, ‘pseudoscientific’ [LeSB14, p.7], and have no verifiable interpretation. That is, the maximum and minimum correlation scores imply neither sufficient nor necessary causation, because having ‘Spell check’ neither guarantees, nor is it required for ‘Writing emails without error’ (Figure 4.4). Neither can the scores be probabilistic causal propositions (as in [PaWe01]), since the ‘needs’ (i.e., causal effects) are vaguely defined without scales of measurement. For example, it is ambiguous whether ‘Write emails without error’ would be satisfied firstly on avoidance of all, most, or some errors, and secondly whether those ‘errors’ are grammatical, lexical, logical, other, or all types. Furthermore, when attempting to understand the semantics of the scores in the cells, it is unclear whether they describe either the likely degree of satisfaction (e.g., ‘some errors’), or the probability for ‘no errors’. Finally, since the scores have no falsifiable meaning (i.e., they represent subjective preferences), it is easy for the practitioner to first decide which product features should be implemented, and then produce a HoQ in order to support that decision, giving only the illusion of a robust decision process.

Overall, the ‘Software HoQ’ does provide a systematic approach through pairwise comparison for explaining the “advantages which customers get” [HeSP03, p.1] from software requirements, described in terms of their estimated ‘correlation’ with the satisfaction of their associated customer requirements. Indeed, experiments on the effectiveness of prioritisation approaches [KBPP12] show that the HoQ compares favourably — bested only by more data-hungry approaches such as TOPSIS [ChHw92]. However, all previous criticisms of subjective value assessments are applicable, e.g., vulnerability to biases, irrefutability, skewed distribution of correlation scores (toward high), etc.

Table 4.5: Feature Comparison Table (Key: ✔ Explicitly, ✅ Somewhat, ✘ Not) Supported

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

Summary of Approach #4: Software House of Quality (HoQ)

**Summary:** A traceability matrix records a ‘correlation score’ between the satisfaction of the solution and the problem. **Pros & Cons:** Allows the modeller to show that a software requirement partially contributes to a user need (and so creates benefit). However, the HoQ’s ‘correlation’ scores are ambiguous, non-falsifiable, and easily affected by biases.
4.4.3 Quality by Design (QbD)

While the HoQ is considered by many practitioners to be the ‘core aspect’ of Quality Function Deployment [HeMa11] (i.e., ‘design-points’ analysis [HeSP03]), QFD actually proposes the use of several interlinked HoQ-style matrices, where the output of a matrix serves as the input of the next [Akao90], or in other words, where the columns of the first matrix become the rows of the next. Perhaps most well-known for a ‘flow-down of matrices’ is Juran’s Quality by Design (QbD) approach [Jura92], which first contextualises customer requirements by mapping them to the stakeholders desiring them, which can then be mapped to customer market segments [HeMa11], and then maps the customer requirements down toward capabilities, components, and qualities of the system.

Crucially, QbD considers customer requirements as means as well as ends, by rating their correlation to the achievement of higher desires. An example of this is Stenning’s proposed QbD chain for predictive evaluation of IT systems (i.e., proposed software capabilities are evaluated before implementation). Business objectives are mapped to project objectives, which are mapped to business process characteristics, which are then mapped to IT system characteristics [Sten99, p.27]. Such a QbD chain provides a ‘chain of argument’ for the importance of requirements via ‘multi-level’ predictive (and later, retrospective) evaluations [Sten99, p.4]. This chaining of HoQ-style matrices could be considered as an exploration of the benefits of a software requirement to a business through a pre-set number of levels of instrumental value generation.

However, chains of means and ends (i.e., predictive causes and effects) in the form of interrelated 2d matrices can be difficult to comprehend, since there is no visual relationship between the means or ends throughout the various matrices. Indeed, Stenning considers any more than five QbD levels unwieldy, but any less than three to provide insufficient context [Sten99, p.26]. Furthermore, chains of instrumental value derivation from a requirement’s source of intrinsic value will vary in length among different requirements, depending on how complex (nested) the business process that the requirement influences is (e.g., where a requirement affects a process that is a sub process of another). This variability in the number of QbD levels needed in order to explore the value of a set of requirements mandates the use of ‘artificial’ levels [Sten99, p.26] that have no real-world correspondence, such as ‘Business Objectives – Abstraction Level 2’. However, this breaches the guideline that each QbD level should be associated with a real-world entity [Sten99, p.26], which was designed to prevent the corresponding HoQ’s from becoming sparsely populated, as well as confusion about what exactly the corresponding HoQ’s are relating. As a consequence of there often being no single and distinct ‘real-world entity’ assignable to each link in a chain of instrumental value derivation (each means→end link), either links would be omitted (causing a fragmented description of the requirement’s value), or artificial QbD levels would prevail.

Finally, the aforementioned weaknesses of the HoQ are unaddressed or exacerbated by QbD: the poor saleability of pairwise comparisons (i.e., every row by every column) is worsened by adding extra matrices, ambiguity in the correlation ratings remains, and explicit support for assigning software requirements to agents (e.g., software components) for planning and traceability, is lacking.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>❌</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Summary of Approach #5: Quality by Design (Chains of HoQ’s)**

**Summary:** Similar to the House of Quality matrix, but chains multiple matrices together such that the entities on the x-axis in the first matrix become the y-axis in the next matrix (i.e., desired ends become means-to-ends).

**Pros & Cons:** Allows the benefit of a software requirement to be contextualised by understanding its contribution to ‘higher level’ benefits; However, chains of instrumental value do not exist in fixed lengths (as is required by QbD).

### 4.4.4 Automated Requirements Traceability Link Generation

Since it is widely recognised that constructing traceability can be very time consuming [ArRi05], numerous approaches have been proposed to automatically create traceability links between requirements using Natural Language Processing techniques. The works of Natt och Dag et al. [NRGB05] and Cleland-Huang et al. [CSRB07] are especially relevant to this thesis, since both aim to suggest links between requirements at different levels of problem-solution abstraction (among other requirement-artefact relationships). (In Cleland Huang et al.’s work, links are proposed between ‘User Requirements’ (problems) and ‘System Requirements’ (solutions), while in Natt och Dag et al.’s work, links are proposed between ‘market requirements’ (‘raw’ customer’s language) and ‘business requirements’ (software company’s language).) While these approaches do not propose a new way of modelling or representing traceability (or indeed a representation of a requirement’s benefit), they offer the potential to significantly ease the process of eliciting the benefits of a requirement, by re-using requirements traceability from previous projects.

However, both approaches generate candidate traceability links between requirements based on the assumption that requirements containing similar words are likely to be related. Conversely, according to the various RE standards and guidelines [Ieee11b, KoSo98, Kovi98, RoRo12b, SaRo09, SoSa97a], there should not be a high degree of similarity between requirements at different levels of problem-solution space abstraction [ELDK14], in order to avoid the redundancy defect [Ieee98]. Hence, the textual similarity analysis techniques employed to generate candidate links would not likely generate useful links between the majority of standards-compliant software requirements and their stated benefits. There are of course rare exceptions where the stated benefits share terminology with the capability or quality, e.g., ‘...should have a light bulb’ and ‘increased ambient light’. But in most cases, automated traceability between requirements at different levels of problem-solution space abstraction would require ‘problem→solution’ knowledge [ELDK14]. This could be inferred from traceability links in previous projects, and so the Natural Language Processing techniques...
used by these automated traceability approaches would rather be most useful for matching similar requirements from other projects, so that these ‘problem→solution’ links can be retrieved.

Finally, while techniques such as Natt och Dag et al.’s can retrieve (or cluster [FeGT13]) similar requirements, they do not make use of the retrievals to recommend links to other elements, as in item-based [LiSY03] (or more broadly, content-based [MeSi10, p.3]) ‘document’ recommendation systems [AJDE07]. Hence, while they would be able to show that ‘SoftwareFeatureX’ in *TheCurrentProject* is similar to ‘SoftwareFeatureY’ in *ProjectB*, they would not suggest to the modeller to consider ‘ExpectedBenefitA’ or ‘ObstacleA’ that were linked to ‘SoftwareFeatureY’. Such a capability is desirable for modelling the value of software requirements, given Chapter 2 (2.1.2)’s conclusion that “…predicting chains of instrumental value generation requires a historic knowledgebase, if they are to be anything more than personal theories about the application domain”. (Generally, the usefulness of recommendation systems for re-using previous project data in software engineering is under-researched, with most focus on the re-use of code or test cases [CaCl10]. In the RE context, success has been found in recommending previously elicited stakeholders [Lim10, p.116].)

Table 4.7: Feature Comparison Table (Key: {✔️ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✖</td>
<td>✖</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>

**Summary**: Has potential to assist the elicitation of a requirement’s benefit (as well as links to other elements such as assumptions, agents, etc.) via the suggestion of candidate links based on traceability from previous projects.

**Pros & Cons**: Assisting the elicitation of a requirement’s benefits improves usability, but its usefulness depends on the quality of the past projects’ documentation. Candidate links may not be applicable to the new project context.

**4.5 Goal Oriented Requirements Engineering (GORE) Approaches**

The requirements engineering community has settled on the concept of goals connected via AND/OR graphs for modelling ‘why?’ a system and its composite capabilities and qualities [Lams01] should be (or have been) chosen. In GORE, a software requirement is considered to be a goal that one software agent (component) is capable of, and will ultimately be responsible for fulfilling [Lams09a, p.19]. Despite GORE’s focus on exploring and recording design rationale as opposed to benefits (which is described in the discussion of Volere’s rationale field), GORE’s techniques and metamodels are highly pertinent to representing the benefits of requirements.
4.5.1 i*

Yu’s agent and goal modelling framework views systems as sociotechnical networks of actors who depend on each other to achieve goals [Yu95]. In order to achieve those goals, actors need to perform tasks and obtain resources, while optimally satisficing associated qualities (soft goals). When using i* for RE [Yu97], a software requirement can be thought of as a goal that one or more actors depend on (i.e., desire for [Yu95, p.39]) a software actor to fulfil. Hence, the benefit of a software requirement can be understood firstly in terms of the actors who depend on it, and secondly for the reasons that the actors have for desiring it. These two concerns are explored by two graph based (nodes and edges) models: the Strategic Dependency (SD) model, and the Strategic Rationale (SR) model.

The Strategic Dependency (SD) model represents inter-dependencies among actors (including humans and software) for the satisfaction of their goals. Dependencies are defined as triples: ‘dependee (actor) → dependum (goal/task/resource) → dependant (actor)’. For example, a ‘weather reporter’ may desire ‘forecasts be accurate’, from the ‘Weather Forecast Software’, which as a result of the dependency link is said to be causally (rather than consequentially [Somm07a]) responsible for fulfilling the goal. To be able to further contextualise the ‘who’ dimension of a requirement, Yu recommends that modelling the generic ‘actor’ class should be avoided in favour of its specialisations: agents, positions, and roles [Yu95, p.24]. For example, an agent (e.g., ‘Doctor John’) can occupy a position (e.g., ‘Physician’), and a position can cover a role (e.g., ‘Treating Patient’) [Yu95, p.24] (and so it can be inferred that an agent plays a role). Through these links, the agents who would benefit from a software requirement could be queried via, for example, the roles that depend on a goal to be satisfied by the software.

The second i* model is the Strategic Rationale (SR) model, which describes the reasoning behind an actors’ dependencies on other actors for their desired goals. SR diagrams are drawn within SD diagrams, inside ‘actor boundary’ elements (which signify an actor’s desire for or ‘ownership’ of the contained elements [Yu95, p.12]). A useful simile for understanding SR diagrams is that an SD diagram’s actors (circles) are like stickmen, and SR diagrams are drawn inside their heads to provide a ‘glass box’ view of their intentions and beliefs [MoHM10, p.163]. Strategic rationale models consist of goal, task, softgoal, and resource elements [Yu95, p.6]. These concepts have important implications for the way in which the benefit of a requirement is considered:

(Hard) Goals are outcomes (desired states in the world) without restriction on how they are fulfilled [Yu95, p.14], e.g., a patient depends on a physician to have their sickness cured, and the physician decides how. Hence, the benefit of a requirement specified as a goal would focus on the outcome, e.g., of not being sick (being cured) compared to being sick (not cured).

Tasks are distinct from goals in that they are particular ways of achieving outcomes [Yu95, p.33], constraining the actor’s freedom in how to fulfil them [Yu95, p.14], e.g., a physician tells a patient to take medication four times a day, two hours after meals. Yu stresses that task specifications need not describe complete knowhow for performing activities. However, the ‘particularity’ of the specified ‘way’ entirely depends on the ‘knowhow’ owned by the evaluator. In the early phases of RE, stakeholders desire goals, but after analysis of the problem and evaluation of alternatives [Ieee11b, p.72], they may gain preference to implementation-specific means (tasks). Finally, the benefit of a requirement viewed as a task would focus on the advantage that the particular means to the end has over alternative means, e.g., taking a different medication, taking it three times a day, etc.

Softgoals are described by Yu as goals without clear-cut satisfaction criteria [Yu95, p.14], e.g., ‘medicine should be taken shortly after meals’. The literature has generally taken ‘softgoal’ to mean
purely this, e.g., in [CIDP05, DKPW07, Liu12]. Even an ontological comparison of the GORE approaches concluded this to be the only difference with hard goals [MaHO07]. However, Yu first described softgoals as the element of choice for modelling Non-Functional Requirements (NFRs) (qualities) [Yu95, p.13], which at first seems to contradict RE best practice that NFRs should be measurable and specific [Maid06], as well as business best practice that goals should be ‘SMART’ (Specific, Measurable, etc.) [CrLy88]. However, i* is described as applicable for early-phase RE [Yu97], where there is an abundance of softgoals over “black and white” “the world is in the stated condition or not” [Yu95, p.14] type goals. In i*, softgoals are said to be implicitly clarified between actors (the depender and dependee) when trying to achieve the goal, rather than there being a prior agreement on their meaning [Yu95, p.14]. This clarification occurs through the identification of alternatives using the dependee’s knowhow, and then through the (undocumented) exploration of reasons for and against them, with the ultimate decision from the depender [Yu95, p.15]. As such, in i*, a goal’s softness is not a temporary state to be later hardened, e.g., through quantification.

However, at the time of trying to implement a stakeholder’s goal, it may be too late to gather their input (which is a well-known cause of software defects [AACF12, AbPa13]). Hence, treating NFRs as softgoals is known to add ambiguity to requirements specifications [DKPW07, p.4], to render impact analysis [Gilb05a] on other goals vague, and to hinder system evaluation (due to the non-verifiability of goals [Sten99]). The latter is due to the fact that the alternatives for satisfying a goal will not be implemented in the same project (and so their relative advantages and disadvantages cannot be verified), whereas measurable quality criteria can be. Indeed, studies on performing summative evaluations of software systems using i* have shown that soft goals lead to “different stakeholders having different understandings, or even no understanding” [LMDL10] of their meaning. Such difficulties are perhaps more likely to occur in predictive evaluations (pre-development) of a requirement’s benefit, since the stakeholders would not be as familiar with the system and its goals.

**Beliefs** (or ‘rationale elements’) belong to actors and provide evidence supporting the arguments for alternatives in a means-end hierarchy [Yu95, p.36]. Beliefs can be used to model ‘applicability conditions’ [Yu95, p.48] to describe the necessary context for the viability of a means-end link. Hence, within i*, an assumption is defined as a belief whose believability is a concern [Yu95, p.49]. As such, assumptions made in claiming some degree of benefit from a requirement can be made explicit and analysed for ‘believability’ [Yu95, p.48], (e.g., that demand for the output of the software functionality will not decline). However, beliefs are not directly part of a means-end hierarchy [Yu95, p.36], and the argument behind the association of an actor to a belief can be unclear [LMER11, p.229].

**Resources** are physical or informational entities required to achieve goals or perform tasks [Yu95, p.15]. For a requirement to realise its benefits, many resources will be assumed to be available. However, usability studies have shown a conceptual overlap between i*’s resources/beliefs/tasks/goals, which complicates choosing an element to model assumptions [ERPM06], e.g., ‘printer paper will be available’ could be modelled as any of those elements with slight adjustment (e.g., as a resource: ‘printer paper’). Finally, risk assessment techniques such as HAZard and OPerability Studies (HAZOPS), among others [AsGM11], have been fruitfully applied to i* models, e.g., to analyse the risk entailed by resource dependencies [ARLM13].

Finally, there are two types of link between elements in i* SR models [Yu95, p.33], derived from the ‘problem reduction’ technique (AND/OR graphs) in the Artificial Intelligence field [Nils71]:

- **Decomposition** links represent the subgoals/tasks/resources required for a goal/task, where all decompositions are necessary in combination to be sufficient for the decomposed. (Both ‘necessity’ and ‘sufficiency’ are for individual goal instances, not the ‘population’ [KrLa92]
of possible instances: e.g., in different projects, a refined goal may no longer be ‘necessary’.)

- **Means-ends** links allow for the representation of the many ways ‘how’ goals can be achieved by subgoals/tasks, and hence ‘why’ those subgoals/tasks should be achieved (where each means is wholly sufficient for the end, and its selection is decided upon design time, except in RE for self-adaptive systems, where decisions occur at run time [CPYZ11, HSMK09]).

Since the focus of i*’s means-ends links (and their equivalent in other mainstream GORE approaches [AGHM10, p.2]) is on describing the rationale for deciding on alternative means, means-ends links are not used to show instrumental value creation in chains of cause-effect (i.e., until the ‘root cause’ or intrinsic source of the value is found). Rather, modellers are to refine new goals/tasks/resources down from those that actors depend on each other for. Hence, the initial selection of the root (highest level) goals is of great importance to the pertinence of all other goals. Indeed, numerous researchers have found this to be a significant weakness of the main GORE approaches [CuPa12, ReWe05, ZaJa97]. Experience reports have also shown that i*’s lack of hierarchy from even the ‘top level project goals’ hinders an understanding of their importance [LMDL10, p.69].

When an SD model is enriched with SR model(s), dependency links can be formed between elements within actor boundaries (i.e., on refined goals/tasks/resources), rather than just between actors. In such cases, the actor-boundary-traversing dependency links imply either goal/task sufficiency or resource necessity. For example, in Figure 4.5, the topmost dependency link in dependency quadruple [#1] can be interpreted as a means-ends link\(^\text{22}\) (sufficiency), and both dependency links in dependency quintuple [#2] can be interpreted as a decomposition links (necessity) [Yu95, p.35].

![Figure 4.5: Three Strategic Rationale Models within the actor boundaries (large encapsulating dashed yellow circles) of a Strategic Dependency Model (adapted from [ERPM06, p.525]).](image)

In real-world projects, the intertwining of SD and SR models can cause considerable usability issues [MoHM10], as well as redundancies and inconsistencies between them [ERPM06, p.524]. Indeed, due to i*’s poor comprehensibility (i.e., lack of awareness and non-obviousness of the i* language [MoHM10]), Ncube et al. propose an approach to translate i* SD models into Volere textual

\(^{22}\) Except, explicitly modelled means-ends links between goals of different actors are not permitted in i* [LMER11].
requirements [NcLM07]. For example, the bottommost dependency link in quadruple [#1] in Figure 4.5 would be translated into numerous Volere shells, such as ‘The <Bank> shall <obtain Bank credit reference>’ (functional requirement), as well as ‘The <Bank> shall be available to the <Rental Car Company> to <obtain Bank credit reference>’ (non-functional availability requirement). For a software requirements specification (rather than a system requirements specification), the textual requirements would be generated only for ‘New Software Actors’, rather than for ‘Stakeholder Actors’ and ‘Adjacent System Actors’ [NcLM07]. Despite it being empirically unclear how advantageous it is to translate a visual model into many more pages of ‘shall do’ text (nor how to translate strategic rationale model elements such as means-ends links), it clearly exemplifies how i* strategic dependency models relate to textual software requirements such as those in Volere (4.3.1).

Finally, to revisit the problem set by this thesis, we are interested in understanding how a requirement, such as Figure 4.5’s ‘The <Bank> shall <obtain Bank credit reference>’ would benefit the stakeholders, such that we can answer ‘why should it be fulfilled?’ and ‘what happens if it is not fulfilled?’. A typical i* model, as in Figure 4.5, is not built to adequately answer such questions, instead merely showing necessity in terms of the process (e.g., for renting a car), rather than in terms of the value it generates as either:

- an outcome — e.g., How much ‘bad debt’ would be avoided by the ‘bank credit reference’ (based on predictions utilising historical records of unpaid debts)?, what is the context to that cost avoidance (does it prohibit profitable rental)?, and does the overall benefit outweigh the overall cost of obtaining the credit reference in monetary terms, in delaying a customer facing process, and in showing lack of trust to the customer?;

- or an alternative means— e.g., How much ‘bad debt’ would be avoided by obtaining a credit check from the bank, relative to the alternative ways of understanding the financial integrity of a customer?, such as by obtaining a credit reference from another actor (like a credit agency), or by a whole new means, such as by requiring whole up-front payment before the rental.

i*’s lightweight set of means-ends link attributes (or more formally its ‘Softgoal-Task’ link attributes [Yu95, p.34]) are mostly to blame for its inability to provide answers to the above questions; The degree of contribution that a means makes to an end (i.e., a task to a softgoal) is defined only qualitatively in terms of polarity (positive/negative) and sufficiency (whether the means is ‘enough’ to completely bring about a vaguely defined end). Consequently, questions about ‘how much benefit?’ a software requirement would bring are for the most part unanswerable by standard i* models.

### Table 4.8: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

|---------------------|-----------|---------------------|---------------------------|---------------------------|-------------------------|---------------------------|--------------------------------------|--------------------------------------|-------------------------------------|----------------------------|------------------------|---------------------|---------------------------|-----------------------------|--------------------------------------|--------------------------------------|----------------------------|--------|-------------------------|------------------------|----------------|------------------------|------------------------|------------------------|-------------------------|------------------------|------------------------|------------------------|
4.5.2 GRL (i* + NFR)

The Goal-oriented Requirement Language (GRL) is described “as a language for supporting goal-oriented modelling and reasoning of requirements, especially for dealing with non-functional requirements” [Liu12]. GRL’s ontology “provides constructs for expressing various types of concepts that appear during the requirement [sic] process” [Liu12]. GRL is an evolution of Tropos [BPWG04], integrating the core elements of Yu’s i* [Yu95] and Chung et al.’s Non-functional Requirements (NFR) frameworks [CNYM99] for modelling design rationale as well as the intentions of a system’s actors [AGHM10, Itu12]. Hence, Reggev and Wegmann consider GRL to be “representative of this [i.e., i*’s] family of methods” [ReWe05]. Finally, GRL is part of International Telecommunication Union Standard (Z.151 [Itu12]) ‘User Requirements Notation (URN)’, where URN’s other component is the Use Case Map notation, which models temporal events, such as the sequenced user and software tasks that fulfill goals.

The main differences between GRL and i* that are pertinent to this thesis are as follows:

1. GRL has support for quantitative as well as qualitative goal contribution attributes [AGHM10], where the latter is mostly inherited from the NFR approach [ChPr09].
2. GRL is more relaxed about how elements can be linked with each other in order to support the wide variety of ways people create goal models [HEAY08], e.g., a ‘contribution’ link is no longer restricted to softgoal destinations [AHGM09, p.6].
3. Intentional elements (goals, softgoals, tasks, beliefs, and resources) need not reside within an actor boundary [AHGM09, p.3] (permitting simpler KAOS-style goal graphs [Lams09a]), since they may not be worth the visual expense when visualising means-ends chains (but actors still ‘own’ the elements in the underlying model).
4. GRL does not make a distinction between i*’s Strategic Dependency (SD) and Strategic Rationale (SR) models, opting for one integrated model with multiple views (diagrams) [AHGM09]. This seems logical since both SD & SR concerns tended to be modelled in an expanded i* SD model [ERP06].
5. Beliefs can be directly linked to means-end links in GRL [Liu12] (where they are not directly part of means-end hierarchy in i*), but this has shown to significantly obfuscate GRL diagrams where many assumptions are modelled [LMER11].

GRL’s contribution links model that softgoals, tasks, beliefs, or links contribute to (influence) each other (Figure 4.6), and are described in terms of the dimensions: polarity, sufficiency, and necessity. Rather than individually describing each dimension for each contribution link, GRL has a set of ‘labels’ (inherited from NFR [ChPr09]) that correspond to values for these dimensions. These labels are listed in order of their desirability (of course, assuming that the link’s destination is desirable):

1. **Positive Contribution**: ‘Make’ is fully sufficient, ‘SomePositive’ provides a contribution of ‘unknown extent’, ‘Help’ is not sufficient, ‘AND’ is necessary, ‘OR’ is sufficient but optional;
2. **Equal Contribution**: ‘Equal’ contributes both positively and negatively.
3. **Negative Contribution:** ‘Hurt’ is not sufficient, ‘SomeNegative’ makes a contribution of ‘unknown extent’, and ‘Break’ is sufficient;

4. **Unknown Contribution:** ‘Unknown’ is a contribution of some kind;

Figure 4.6: GRL Goal Graph providing the rationale for the contribution link between ‘VoiceLan’ to ‘Transmit Voice’ using a belief element (GRL diagram modified from GRL’s ontology [Liu12])

The difference between GRL’s ‘Means-ends’ and ‘Contribution’ links is disputed, since a ‘Make’ contribution link seems semantically identical to a ‘Means-ends’ link. To address this, Guizzardi et al. propose that the distinction is based on the notion of intention. E.g., modelling $X \rightarrow Y$ with a means-ends link implies that it was X’s ‘deliberate’ intention to achieve Y, while if it were modelled with a contribution link it was not X’s intention [GFGW13]. While this seems logical with their contrived toy example using an undesirable goal (‘take a car sick pill’ is related to ‘asleep fallen’ with a ‘Make’ contribution, and to ‘car sickness prevented’ with a ‘Means-end’ contribution), it does not apply in many other cases (e.g., in Figure 4.6 for link [A]). Contrary to Guizzardi et al.’s conclusion (and more inline this this researcher’s understanding), numerous researchers previously concluded that means-ends links and contribution links are essentially identical, except for that the latter describes polarity and degrees of sufficiency [EnWi12, p.310]. Furthermore, the ability to model that X influences Y “without explicit desire”, is already addressed by GRL’s ‘Correlation’ link [Liu12].

In order to contextualise softgoals, ‘Softgoal Interdependency Graphs’ (as inherited from the NFR approach [MyCN92]) decompose abstract qualities into their components using ‘AND’ contribution links. Then, the influence that softgoals have on each other is described, as shown in Figure 4.7. However, due to the qualitative descriptions of softgoals and contribution, it is not possible to unambiguously answer ‘what if?’ and ‘what if not?’ impact questions for understanding a softgoal’s benefit. For example, contrary to Figure 4.7’s implication, it is possible for there to be good ‘Performance’ without ‘Good System Management’, e.g., disabling security features could increase ‘performance’ and decrease ‘good management’. It is also ambiguous how ‘Good System Management’ contributes positively to ‘Supportiveness’, since it is left to the model reader’s interpretation both:

- the observable variable(s) influenced by ‘Supportiveness’ (i.e., what exactly does it refer to?);
- the direction of desired influence on those variables — that is, is it good for ‘Supportiveness’ to be low or high? (i.e., the “acceptability direction” in the QoS ontology [DoLo05] is unclear).
A strong argument against qualitative contribution scores can be drawn from outside of the GORE literature. Both Wiegers [Wieg03] as well as Egyed and Grünbacher [EgGr04] describe interaction matrices for the various generic non-functional requirements. The former’s matrix describes ‘Efficiency’ as having a negative relationship with ‘Usability’, while the latter’s matrix describes a positive relationship. Indeed, both are arguably correct in their unqualified states (i.e., not specific to a subject/object/concept). In the words of Pauli (originally on the falsifiability of scientific hypothesis), the model “is not only not right, it is not even wrong!” [Peie60]. A less ambiguous approach would involve more precise (formal or quantitative) softgoal descriptions, as well as more rigorous causal propositions that avoid reliance on the perceptions of those comprehending the model.

In an effort to improve the precision of contribution links, GRL proposes that contribution can be described quantitatively, where the modeller selects a number from a dimensionless interval scale similar to the scores used by the House of Quality [HaCl88], but using an interval of [-100,100] rather than the HoQ’s [1,9] interval. As further departure from the HoQ’s, instead of the contribution scores describing correlation, GRL’s scores are mapped to the aforementioned qualitative contribution labels, as is illustrated by Figure 4.8. Given that these scores are dimensionless and hence untestable, perhaps the most useful aspect of GRL’s quantitative contribution scores is that values between the otherwise qualitative labels are permitted [AGHM10, p.28]. For example, rather than two contributions being ‘Help’, their slight inequality may be represented by specifying first goal’s contribution as 25 and the second goal’s contribution as 30. (A study comparing by Liaskos et al. [LiHR13] concurs that this is the most significant aspect.)

Perhaps unsurprisingly, the ‘Equal’ contribution label is the only contribution label not mapped to a numerical value. Indeed, it would seem bad modelling practice to describe a contribution as equally good and bad, since the reasons for the bipolarity are left undescribed, which is against RE’s tenet of eliciting and explicating knowledge [PiKa11]. Equal contribution would also indicate that the destination goal is ambiguous, which could lead to poor decisions based on incorrect, hidden, and hence unchallenged beliefs about it. E.g., if ‘Use VoiceLan’ both lowers and increases...
‘costs’ with the same amount of contribution, then an inadequate definition of ‘costs’ is indicated and should be clarified as the costs of the ‘initial purchase’, ‘hardware maintenance’, and so on.

Amyot et al. caution modellers that translating GRL’s qualitative labels into quantitative scores runs the “risk of inserting a precision into our results which does not derive from an authoritative source” [AGHM10, p.28]. However, the risk of inserting false precision still exists with GRL’s quantitative contribution scores, regardless of whether the scores were translated from labels, or chosen first-hand. Contribution scores other than -100, 0, or 100 especially imply a false sense of precision and semantic meaning. For example, in distinguishing the difference between 25 (“positive but not sufficient [extent of the contribution]”) and 75 (“positive, but the extent of the contribution is unknown” [AGHM10]), it appears as though describing opinion on the extent of the contribution is conflated with certainty in that opinion. Indeed, one would assume that the extent of a ‘75’ contribution should be higher than that of a ‘25’ contribution, but instead the only formal difference, as per the previous definitions, is that 75 has an ‘unknown’ extent. (The labels at the bottom of Figure 4.8 seem to specify extents of sufficiency with unspecified certainty, while those at the top specify degrees of certainty in unspecified extents of sufficiency.)

Additionally, as a result of GRL’s bounded [-100,100] contribution interval, it is not possible to model that the satisfaction of a goal would exceed the satisfaction of another, such as an estimated 150% increase in revenue as opposed to the (evidently pessimistically) desired 50%. Finally, using 0 to represent ‘unknown extent’ could be incorrectly interpreted as ‘no contribution’. (In computer programming, ‘null’ is used rather than 0 or ‘’ for this reason [SeHa11, p.8].)

(More advanced approaches for quantifying qualitative contribution and then determining the qualitative degree of goal satisfaction are proposed in the literature, e.g., by using fuzzy logic to describe contribution labels as membership functions [SeSa11, TCKQ12]. However, by Serrano et al.’s own admission, the results (i.e., the calculated degrees of consequential goal satisfaction) vary depending on the choice of defuzzification algorithm. Furthermore, many criticisms of qualitative contribution are unaddressed, since the contribution label membership functions are not rooted in application domain phenomena, and so are unverifiable. Finally, statisticians consider fuzzy logic inferior to probability distributions since it “enshrine[s] and celebrate[s] people’s imperfections” rather than embracing unavoidable uncertainty [OhOa04].)

Lastly, GRL’s ontology stipulates that a ‘contributee’ (the destination of a contribution link, e.g., ‘Lower Costs’ in Figure 4.6) should be a either a softgoal, another link, or a belief (but not a hardgoal), despite Amyot’s previously mentioned claim that GRL is less constrained than i* [AHGM09, p.6]. Hence, the vast majority of GRL models in practice,—especially those created with tools enforcing its metamodel, have contribution links going only toward ambiguous softgoals. Since softgoals have no defined satisfaction criteria, there cannot exist unambiguous descriptions about the extent to which a softgoal’s satisfaction would be contributed toward, since what it means for the softgoal to be satisfied is undefined.

Overall, the NFR framework’s focus on ‘satisficing’ softgoals (inherited from Nilsonn’s work in Artificial Intelligence [Nils71], and then inherited by GRL) rather than satisfying ‘hard’ goals is considered the primary source of this ambiguity. In the field of economic decision analysis, Schoemaker succinctly summarises that “satisficing is just a more general type of optimizing, including such factors as the cost of information, decision time, constraints, and cognitive effort. It is the latter degrees of freedom, however, that may make the … approach tautological: i.e., non-empirical and non-falsifiable”[Scho82]. Within RE, it is generally acknowledged that “a description that is not refutable [i.e. falsifiable] doesn’t say much”[Jack95a] — as inspired by Popper’s argument
that any hypothesis is meaningless without it being falsifiable (as opposed to verifiable) [Popp63]. This thesis argues that predictions about the benefits of a requirement should be falsifiable hypothesis to be rejected or accepted, otherwise they are no different to a soothsayer’s vague predictions — correct but useless [Popp63]. To paraphrase Jackson [Jack95b], without refutability there is always the evasion tactic of, for example, “Well, it depends on what you mean by performance, and what you mean by fast”. This thesis’ position on softgoals cannot be better summarised than paraphrasing von Neumann: “There is no sense in being precise [about the benefits of a requirement] when you don’t know what you are talking about [i.e., what the requirement/benefit refers to]”. A parallel can be drawn to the decision analysis field, where inferior (i.e., qualitative) processes have found favour with decision makers because “they do not force you to think very hard”; “In decision-making, as in many other pursuits, you have a choice of doing something the easy way or the right way, and you will reap the consequences” [Howa07].

In summary, GRL (like other goal modelling approaches) deals well with externalising knowledge that is otherwise personal, subjective and situation dependent [Nona94]. However, externalising stakeholders’ mental models in qualitative (or quantitative with qualitative meaning) models is not normally sufficient for achieving complete and common stakeholder agreement and understanding, neither on a conceptual model of the problem to be solved, nor on the optimal system response to it [KaLo03]. In other words, the absence of “parameters, inputs, initial conditions and generally of factors that are needed for testing these qualitative models” [KaLo03] diminishes the information value of GRL’s models, since stakeholders may experience difficulties in comprehending or ‘testing’ the implications of changes to system components on the overall behaviour and state of goal satisfaction, even for the simplest of systems [LoZP03].

Table 4.9: Feature Comparison Table (Key: {✔️ Explicitly, ✅ Somewhat, ❌ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>❌</td>
<td>✔️</td>
<td>❌</td>
<td>❌</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>

Summary of Approach #8: Goal-oriented Requirement Language (i* + the NFR Framework)

**Summary:** Similar to i*, but adds the capability to qualitatively (or pseudo-quantitatively) describe how a software requirement can partially satisfy (i.e., contributes to) another goal (and therefore create some benefit).

**Pros & Cons:** Little effort is required to postulate qualitative theories about the contribution a requirement makes to a goal, so the approach has high usability. However, the resulting non-refutable propositions can be meaningless.

4.5.3 NFR Framework + Goals Questions Metrics (GQM)

Daneva et al. aim to reduce ambiguity caused by softgoals in the Non-Functional Requirements (NFR) framework, by combining NFR with GQM (Goal-Questions-Metrics) in order to ‘harden’
the most important softgoals [DKPW07]. Chung’s NFR framework specifies non-functional quality requirements as softgoals comprised of a topic and a quality (i.e., the softgoal syntax: ‘Quality[Topic]’ [Chun07, p.15]). Basili’s GQM [Basi93b] supports the identification of metrics from goals by refining measurement goals into questions, whose answers require metrics, collectively representing an operational and unambiguous definition of the goal. As a result of Daneva et al.’s NFR and GQM integration, two new types of goal are proposed for the NFR framework: an ‘NFR hardgoal’ (quantified), and an ‘Operationalizing NFR hardgoal’ (a functional solution to the NFR).

For example, the ‘Performance[Account]’ softgoal (topmost in Figure 4.9) is considered to be highly important to the stakeholders, and so it is ‘AND’ decomposed into a ‘hardgoal’ (rightmost middle in Figure 4.9) using GQM, which should be satisfied rather than satisficed (i.e., ‘good enough’ [ChPr09]).

Figure 4.9: Using GQM to make softgoals 'hard' in NFR [DKPW07]

However, unambiguously ascertaining the degree of sufficiency for an NFR hardgoal that an ‘Operationalizing NFR hardgoal’ contributes it is not possible, due to NFR’s qualitative contribution labels (discussed in GRL’s critique 4.5.2). Furthermore, the rationale for the NFR hardgoal’s prescribed target (‘\( \leq 3 \) seconds’) is not described by the approach. That is to say that the hardgoal itself is not treated as a means to an end, but rather as a refinement of the softgoal’s definition.

Daneva et al.’s understanding of the distinction between softgoals and hardgoals provides a new perspective on GRL’s softgoals: Softgoals can be considered as those that the stakeholders (or less desirably, the architects [AACF12]) do not consider to be worth the effort to harden. This view that not all goals need to be stated precisely (i.e., NFR’s “no quantification, no existence” anti-pattern [Chun07, p.18]) is held by numerous advocates of quantification in software engineering, e.g., “If you cannot put numbers on your critical system variables, then you cannot expect to communicate about them, or to control them” [Gilb05b]. Conveniently, in NFR, each softgoal can be associated with a criticality value of \{extremely critical, critical, non-critical\} [ChPr09, p.372]. However, determining the criticality of a goal before its contributions to others are modelled seems slightly tautological. Furthermore, this qualitative approach for weighting the relative importance of goals is similar to the MoSCoW priorities, and so the critiques on its ambiguity in Section 4.3.1 apply.
Table 4.10: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of Approach #9: The NFR Framework + Goals Questions Metrics (GQM)

**Summary:** Adds to GRL (through its inheritance of NFR) the capability to describe a ‘softgoal’ as an ‘NFR hardgoal’ (i.e. a quantified non-functional), but allows qualitative softgoals to coexist if they have low priority/impact on the software.

**Pros & Cons:** Makes non-functional requirements measurable, topic-specific, & qualitatively contributable to by functions. However, contribution is ambiguous & ignorant of uncertainty, and the NFR as a means to another benefit is not described.

### 4.5.4 KAOS + Objective Gauge Variables

The first GORE language, KAOS (Keep All Objectives Satisfied), was defined in 1991 [DaFV91], pre-dating i* [Yu95], NFR [MyCN92], GRL [AHGM09], GBRAM [AnPo98], GSN [AtKM04], Tropos [BPGG04], and so on [YGMM10, p.577]. As opposed to the other qualitative and early-phase RE oriented GORE languages, KAOS was designed for more formal, later-phase RE. For the purpose of this thesis, the various concepts within the GORE languages are considered interchangeable, and indeed, van Lamsweerde’s textbook [Lams09a] shows the various concepts intertwined in a language-agnostic manner. However, two notable differences between KAOS and i*/GRL are that KAOS allows for alternative agent refinement as well as goal refinement, and that KAOS has methodological support for deciding among alternative refinements, accounting for the ‘weight’ of the various goals.

Firstly, the rightmost section of Figure 4.10 shows KAOS’s support for modelling alternative goal-agent responsibilities within one model. This is contrary to i*’s use of two separate strategic dependency models: one showing actor responsibilities in the as-is system, and another for the to-be. This is pertinent to this thesis since benefit is realised in many IT systems by transferring causal responsibility for goal satisfaction from one agent (human) to another — either to a more affordable human (enabled by the software), or to software. Hence, the ‘do nothing’ option, which provides a baseline for any proposed benefit, could be represented by the as-is agent-goal responsibility link.

![Figure 4.10: KAOS alternative agent refinement (right), as well as goal refinement (left) [Lams09a]](image-url)
Secondly, van Lamsweerde proposes an approach to evaluate (and hence decide on) alternative functional requirements (i.e., leaf goals) in terms of their contribution to ‘quality’ softgoals [Lams09b], based on the third way (identified by van Lamsweerde) of describing contribution between goals (where the first two are similar to i*/GRL’s (Section 4.5.2), and the third to NFR+GQM’s (Section 4.5.3):

1. using subjective qualitative labels i.e., estimated as one of {−−, −, +, ++};
2. using subjective quantified scores i.e., estimated as one of [−100, 100];
3. using objective ‘gauge variables’, e.g., estimated as one of [1, 99] ‘seconds response time’.

van Lamsweerde concludes that third approach is the most appropriate for deciding among alternative requirements, for the transparency and objectivity of the resulting conclusion [Lams09b]. (This concurs with the conclusions reached throughout Chapter 2’s conceptual framework.)

However, these approaches treat leaf soft goals as intrinsically valuable, given that goal contributions are estimated only between two goal refinement levels: one leaf behavioural goal and one quality leaf soft goal24. As such, the contribution that each softgoal makes to higher level stakeholder goals is not estimated, and so knowing the benefit of ‘minimising participant interaction’ (to use an example from the paper), e.g., from 4 interactions to 2, is not possible. Instead, it is assumed that the elicitation of softgoals via goal graph refinement ensures their pertinence, but the starting point is ‘system be effective’, rather than from stakeholder problems. Hence, the softgoals may be based on preconceived qualities of software systems rather than on the stakeholders’ problems, which might entail intervention outside of the system-to-be’s scope in order for the software’s expected benefits to be realised. Indeed, KAOS goal models are bounded by the capabilities of the agents within the scope of the software system [Lams09a, p.317], and so indirect benefits may not be modelled.

Furthermore, estimates on the contribution that an alternative requirement makes to a leaf soft goal’s gauge variable are specified using ‘=’ arithmetic, rather than ‘+’ or ‘−’, and so the contribution of requirements contributing to the same benefit would be overwritten, instead of cumulatively combined. (It is important to clarify that gauge variables are cumulatively propagated additively up through the softgoal goal graph, e.g., ‘MinimumOverheadOnParticipant’ = ‘MinimumInteraction’ = ‘MinimumReplanning’ + ‘MinimumInteractionForGettingAvailabilityConstraints’, since the softgoal graph’s purpose is to conceptually refine the definitions of the comparison criteria. In other words, this paragraph’s critique applies to the contribution links between alternative software requirements and leaf soft goals, — not between leaf soft goals and soft goals. However, even this technique for propagating up a softgoal graph is not ideal, since “When a value for the variable at some subgoal makes no sense, we just ignore it in the summation” [Lams09b].)

Perhaps the most interesting aspect of this approach (relative to the so(far evaluated approaches) is that all contributions from subgoals to parent goals are normalised. (The normalisation method is to calculate the percentage of a goal’s satisfaction due to its contributions, relative to its ‘target’ level, and then to multiply this percentage by the goal’s relative weight of importance, where all weights sum to 1). Because of this, the benefit generated by one software requirement could be compared to that of another, using the same comparison scale (theoretical implications of normalisation aside [HeBa06]), even where they belong to a different (i.e., orthogonal) goal refinement branch. Despite the weightings of a goals’ inward links being subjective, the softgoal gauge variable estimates affected by the weights take as inputs verifiable and objective metrics. Besides, when it

23 A more comprehensive comparison of the various qualitative goal contribution reasoning techniques can be found in [HoYu12] (they are not covered here in depth due to the ambiguity of qualitative reasoning).

24 An analogy is that one ‘House of Quality’ is constructed (Section 4.4.2), rather than ‘Quality by Design’s many inter-linked HoQ’s (Section 4.4.3), and so any benefits would not be contextualised.
comes to comparing goals on completely different scales on different goal refinement branches (e.g., not even comparing apples to oranges, but apples to cars), especially where there is no relation between them, nor a quality for comparison (e.g., amount of vitamin C provided in the former example), then subjective assessments must occur, resting on preferences, utility, indifference, weights, and so on — as in this approach. Indeed, in order to study the behaviour of a consumer (i.e., why someone has preference over alternatives), it is ultimately “necessary to have the knowledge of [subjective] utility from the consumption of a good” [Jain08, p.99]. However, the model’s quality afforded by the objective metrics can easily be undermined by this subjective process of utility and weight scoring, as exemplified by the bias found in studies of a similar technique in the context of employee performance measurement [ItLM03].

Finally, KAOS’s language includes concepts not found in GRL, such as obstacles to goals (in this context, to realising benefits), explicit responsibility links, and the distinction between software-to-be and environment agents, as well as the requirements or ‘expectations’ on them [Lams09a, p.490].

| Table 4.11: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported) |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|                  | ✔               | ✔               | ✔               | ✔               | ✔               | ✔               | ✘               | ✘               | ✔               | ✔               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               | ✘               |

Summary of Approach #10: KAOS + Objective Gauge Variables

Summary: Describes the benefit of a requirement (leaf behavioural goal) in terms of a degree of satisfaction for a leaf softgoal (a numeric variable to be minimised or maximised). Weights & normalises softgoals for their cross-comparability.

Pros & Cons: Objective gauge variables provide objective (refutable) arguments for a requirement’s benefit. However, only one level of goal contribution is described per requirement, so benefit predictions can be shallow (w.r.t. previous footnote 24).

4.5.5 KAOS + Probabilistic Goal Contribution

In order to be able to specify that a goal’s satisfaction is only required “in X% cases”, or according to some probability distribution or distribution summary (e.g., ‘X mean time between failure’), van Lamsweerde proposes probabilistic goals, which lie between softgoals and hardgoals [Lams09a, p.269]. As such, hard goals can be described to have a ‘partial degree of satisfaction’, which is different from the subjectively determined ‘satisficed softgoal’, in that the number of cases where the goal was satisfied can be objectively determined, hence reducing evaluative ambiguity.

A probabilistic layer for quantifying goal graphs was first proposed in the literature by Letier and van Lasmsweerde [LeLa04]. Their approach is essentially the same as that in the previously discussed ‘KAOS + Objective Gauge Variables’ approach (Section 4.5.4), except for that random variables are used instead of constant numbers when estimating gauge variables. The resulting analysis “is more accurate, but more heavyweight as a price to pay” [Lams09b], and so quantitative
probabilistic targets typically “will be specified on [and hence goal contribution estimated for] important high-level goals of the application domain, and on a few key lower-level goals” [LeLa04].

A brief summary of the approach [LeLa04] is that variance in a behavioural goal’s contribution to a softgoal (i.e., various states of partial, full, or exceeding goal satisfaction) is represented using Probability Density Functions (PDFs) to describe the likelihood of the possible values for the softgoals ‘gauge variable’ given that each alternative is implemented. To use an example from their paper, consider a goal to reduce an emergency services’ average ‘Call Taking Time’ (measured in seconds). Two alternative electronic systems to replace a current paper based system are proposed, hence, there are three probability distribution functions to be defined for the softgoal: one for the current system, and two for the new alternatives. For example, one might state that ‘Call Taking Time’ would follow a normal distribution with the parameters (mean=60, standard deviation=45 seconds) given the implementation of AlternativeSoftwareFeature1. Hence, this approach allows for the description of contribution uncertainty, and hence uncertainty quantification by simulation. (Since the contribution estimates are probabilistic, simulations can be run to better understand the likelihood of the desired benefits being realised, especially where the benefit depends on the satisfaction of many independent goals, e.g., using Monte Carlo simulation [MMAG11, SFBD11] or numerical integration [LeLa04]. However, where more than a few alternative requirements are modelled, even formulating the combinations of requirements (i.e., defining alternative systems) to be simulated becomes challenging due to the exponentially growing possibilities. Hence, simulations for every possible set of requirements can be automated with an exhaustive (‘brute force’) search, or for larger problems, with genetic algorithms [HeLe11].)

However, reasons for the variation in contribution are not described, which would describe why and when a requirement will contribute some benefit, e.g., variation due to environment conditions, as are describable with ‘usage profiles’ [Elli13] or ‘operational profiles’ [Musa93]. Furthermore, epistemic uncertainty (variation reducible by more knowledge [OBDE06, p.10]) is not explicitly described (e.g., that the stakeholder is only 50% confident that the mean call taking time would be 60 seconds). Such uncertainty must exist to some extent, since the estimates are about future events, i.e., after the requirements are implemented, with a different context than from past experiences.

Additionally, effects of the variance on the satisfaction of higher goals (perhaps outside the scope of the system, e.g., business objectives) are not described. To use the example root goal provided in their paper, the effects of an ambulance arriving at a scene within 8, 14, or 16 minutes (i.e., satisfaction of the goal’s target exceedingly, completely, or partially) are not described in the context of the benefits of doing so. If this is not explored, then it might be that there are no significant benefits to be gained at certain intervals of goal satisfaction levels. Hence, if a goal is defined with a specific target (e.g., target arrival time) in mind, without the rationale for doing so explored as further goal abstractions, then satisfying that goal may not be worthwhile, since “wrong decisions may be taken if they are based on wrong objectives” [LeLa04]. Additionally, the approach does not capture stakeholder “attitude, preference and likings” [LiJA12] in the various degrees of softgoal satisfaction (which, as later discussed in Section 4.5.6, are typically described with utility functions).

Finally, mathematically heavy probabilistic approaches such as this, which require the modeller to understand numerical integration, probability distributions, and write simulation computer code (due to its lack of tool support) have limited usability and hence applicability. Even if the modeller and his/her peers understand the model and its conclusions, the typical set of software project stakeholders might have to take the conclusions as expert advice and may not contribute feedback.
Table 4.12: Feature Comparison Table (Key: ✔ Explicitly, ✅ Somewhat, ✘ Not) Supported

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>

Summary of Approach #11: KAOS + Probabilistic Goal Contribution

Summary: Allows the description of contribution between alternative requirements and quality goals in terms of probabilities for each possible value on the quality goal’s ‘gauge variable’s scale (e.g., response time in milliseconds).

Pros & Cons: Allows the modeller to describe (and simulate) variance (aleatoric, not epistemic) in goal contribution. However, it does not propagate degrees of goal satisfaction (means) to other goals (ends) in terms of the end’s scale. Furthermore, the approach has no computer tool support (especially important for simulation), and so usability is poor.

4.5.6 KAOS + Utility Functions

Goal graphs in requirements engineering (and AI — their original field) tend to imply binary success criterion, i.e., “either the goal formula holds at the end of the execution or it does not” [HaHa98]. However, it is important to be able to specify, for example, that <1 minute ‘new account creation’ time is ideal, <2 is preferable, but >2 is useless. In response to this, and to the criticism on ambiguous root softgoal ‘importance’ ratings in the NFR framework (Section 4.5.2), researchers have proposed the description of weighted utility in goal graphs (as is core to ‘Multiple Attribute Utility Theory’ [ChHw92], made popular by behavioural economics and decision theory [Howa07]). The approach chosen as representative of this technique is Fitzgerald et al.’s KAOS + utility integration [FiKH11], while other examples include weighted utility in requirements hierarchies [Bued11], and utility functions on the fit criterion fields of individual non-functional requirements [HuJD11, ReHB07].

Utility functions, as depicted by the graph on the left of Figure 4.11, convert an attribute’s various values (e.g., ‘response time’ measured in milliseconds) into a normalised utility score in the interval [0,1], that represents the ‘goodness’ or ‘acceptability’ of each value to a person [Bued11]. To evaluate the goodness of each decision option, their utility scores for each of the decision attributes are multiplied by the attribute’s weight, and then aggregated (summed) into one final comparable score. Utility functions can either describe aggregated (conflated) attributes (as in [Lock06]), or ‘atomic’ attributes (as in Figure 4.11). In the former case, the utility function can rise and then fall with increases in the attribute’s value, whereas in the latter case, the utility function is monotonic. While the former’s conflation of many attributes requires less effort to describe, the latter describes why the utility is not monotonic (in the spirit of RE), and so is preferable [Bued11]. For example, if the utility function for ‘pints of milk to purchase’ appears non-monotonic, then it should be decomposed to variables such as ‘amount of milk to be consumed’ and ‘amount of gone-off milk to be wasted’.
Figure 4.11 provides an example of a goal graph’s integration with utility functions and weights. (Popular approaches for eliciting utility functions are *Difference Standard Sequence; Bisection;* and customising pre-set curves such as *Linear, Concave, Convex,* and *S-Curve* [Bued11, chap.13]. Decision weight elicitation methods include: *Spread 100 points; Anchoring on the least or most important; Approximation based upon ranks; Analytic Hierarchy Process; Balance Beam; Trade-off; Pricing Out;* and *Graphical Bar Adjustment* [Bued11, p.415]. Weights are distinct from probability, given that they are “aimed at reflecting the impact of events on the overall attractiveness of gambles” [Scho82].)

The top level goal (in the rightmost goal graph) is known in AI as the ‘super goal’ or ‘terminal value’, which in the context of software systems engineering, would be ‘System be good’. Of course, ‘good’ is not freestanding and cannot be left unqualified [Geac56], and so goals entailing the ‘goodness’ of the system are refined from it. This second level of goal refinement can be considered as root goals in the NFR framework (and so GRL). Each link in the goal graph defines its relative importance amongst its siblings (as defined by having a common parent goal). Conveniently, this allows the importance of any goal’s degree of satisfaction to be compared with another. E.g., if the leaf level goal ‘Maximise Fire Rate’ variable ‘Reload Time’ is 1 second, then the goal satisfaction’s relative (comparable) utility score is $0.45 \times 1 \times 0.1 = 0.045$, as defined by: \[ \text{utilityFunction(MaximiseFireRate, 1 second)} \times \text{linkWeight(MaximiseFireRate, MaximiseFrags)} \times \text{linkWeight(MaximiseFrags, MaximiseScore)}. \]

Such an approach is aimed at calculating the optimal configuration of goal satisfaction (i.e., value of variables) to maximise the utility of a system. It is not designed to explore the instrumental value of a requirement, and so contribution links from low level goals to high level goals do not describe how the goal’s satisfaction causes some degree of satisfaction for another goal. Rather, it is assumed that the modeller is able to describe the intrinsic value of any goal variable (e.g., ‘response time’) via the weighted utility of any of the variable’s possible values (without acknowledging uncertainty). (Both Regnell et al. and Hull et al. make this assumption in their approaches for describing the ‘goodness’ of degrees of satisfaction for a non-functional requirement as utility functions [HuJD11, ReHB07].) Finally, this assessment of importance is not placed into the context of its effects on other goals, — e.g., effects of ‘maximise fire rate’ on ‘maximise frags (kills)’ are not estimated.
Table 4.13: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Summary of Approach #12: KAOS + Utility Functions

**Summary:** Adds to GRL the capability to express various degrees of utility for satisfying a goal to different degrees (in terms of the goal’s objective, quantitative scale). ‘Root goals’ are weighted such that all utilities are comparable.

**Pros & Cons:** Provides a mechanism for comparing otherwise incomparable benefits achieved by the satisfaction of a software requirement. However, it does express the benefit in terms of another benefit (i.e., its instrumental value).

4.5.7 KAOS + Confidence Factor Annotation

On the premise that a software project’s success depends on its stakeholders being confident about the requirements set’s quality (completeness etc.) [NuEa00], Boness et al. propose a methodology for assessing stakeholders’ confidence in a KAOS goal graph’s requirements [BoFH11, Bone11]. In essence, the approach elicits six confidence ratings from stakeholders or experts for each individual goal and agent [Bone11, p.56]. These confidence ratings are split over six ‘Confidence Factors’:

1. **Foundation:** the adequacy of root goals for the system’s intention and scope;
2. **Refinement:** the completeness and minimalism (i.e., not gold-plating) of goal refinements;
3. **Assumption:** the soundness of allocated goal-agent responsibilities (i.e., regarding the capabilities of the system’s and the environment’s agents), and of domain assumptions;
4. **Achievability:** the likelihood that the software agents can be built to satisfy the acceptance criteria defined by the leaf goals, within the project’s resources;
5. **Conflict:** the degree of unreconciled conflict between any pairs of goals.
6. **Engagement:** the degree of stakeholder/expert scrutiny on the goal graph.

However, rather than using probabilistic risk metrics, Boness et al. propose to rate these factors using qualitative confidence ratings on an ordinal scale (High, Medium, Low, None [Bone11, p.90]), purportedly avoiding “a level of precision which could not be guaranteed” [BoFH11]. Interestingly, Boness later converts these “subjective risk scales” to numerical “risk ratings” to be later used as probabilities, e.g., ‘Medium’ becomes ‘0.4 ≤ x ≤ 0.7’, which seems to contradictorily imply a level of precision. This exploits that people often prefer providing estimates of likelihood using verbal expressions in contexts ranging from sports [OBDE06, p.86] to nuclear power safety [OBDE06, p.205]. However, an extensive body of estimation studies concludes that “the key messages about verbal expressions of uncertainty are that they mean different things to different people and sometimes mean different things to the same person in different contexts” [OBDE06, p.87]. As such, confidence ratings on the stakeholders’ belief that a requirement will satisfy the stated goal (i.e., the ‘Refinement’ and
‘Achievability’ factors), might be so ambiguous and context-dependant so as to be non-descriptive.

For example, Sutherland et al. report on a study in which the standard deviation of percentage chance estimates assigned to terms ‘unlikely’, ‘rare’, etc. exceeded 30 [Suth91], showing that variation in ratings may be entirely due to different interpretations of the terms for each goal’s context.

Finally, Boness et al.’s ‘Achievability’ confidence factor refers only to the confidence that a stakeholder has in a requirement being fully satisfied, and their ‘Refinement’ factor refers only to whether a refined requirement or its set of siblings would be fully sufficient for its parent goal. However, many requirements partially satisfy goals (Section 4.5.5), often with additively but non-equal contribution. Hence, a stakeholder might have little confidence that a requirement would be fully satisfied, but be highly confident that it would be slightly less than fully satisfied. Along the same lines, confidence in various possible extents of contribution from a goal’s refined requirements are not considered.

Table 4.14: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

Summary of Approach #13: KAOS + Confidence Factor Annotation

Summary: Qualitatively annotates each goal with a stakeholder’s confidence in six factors, of which, the ‘Refinement’ and ‘Achievability’ factors could be used to describe confidence that a requirement is fully sufficient as a means to an end.

Pros & Cons: Qualitative ordinal ratings of likelihood that a requirement will lead to the full satisfaction of its related goal(s) are easy and intuitive to make, but are proven to be non-descriptive and inhibit analytical thinking [WiWe96].

4.5.8 GRL + (Key Performance) Indicators

Pourshahid et al. recently extended the jUCMNav goal modelling tool, which is compliant with GRL’s parent URN standard, to allow the relation of Key Performance Indicators (KPIs) to goals [PAPG09]. Hence, a goal’s degree of satisfaction (derived from its quantitative contribution scores, as discussed in GRL’s Section 4.5.2) can be mapped to real world numbers, as is similar to KAOS’s ‘objective gauge variables’ (discussed in Section 4.5.4). However, ambiguity may still exist in goal chains (i.e., >1 link), since KPIs do not affect the way in which goal contribution is specified further up a goal chain (i.e., as low-level benefits are translated to high-level business objectives, e.g., converting time to cost). Additionally, interaction and relationships between KPIs are not modelled.

Since the publication of this thesis author’s first published work on this topic [ELDH12], Horkoff et al. improved GRL’s integration with indicators to consider the hierarchy of KPIs alongside a goal model [HBJY12]. (As evidence of the pertinence of this author’s paper [ELDH12], Horkoff et al.’s co-authors cited it in a more recent related paper [RaMA13].) However, their approach [HBJY12]
is concerned with improving Business Intelligence (BI) reporting, rather than aligning software requirements with their sources of expected benefit. Thus, numerous areas are lacking when applied to our problem, including that:

- Stakeholder confidence and utility through the range of possible degrees of goal satisfaction (i.e., KPI values) is not specified, making it hard to know the credibility and likelihood of the estimated contribution, as well as effects of partial goal satisfaction;
- Potential variation in goal contribution (i.e., aleatoric uncertainty in the range of possible values for a KPI, given a requirement’s implementation) is not described, as can be achieved with software ‘usage profiles’ (i.e., ‘operational profiles’ [Musa93]), as described in [Elli13];
- Non-linear relationships in the associated KPI hierarchy (i.e., diminishing returns in achieving an objective) are not amenable to algebraic description [Ster00] (i.e., ‘business formulae’, as termed in the approach), complicating their definition, communication, and experimentation;
- Actions (immediate means, e.g., software requirements) are not distinguished from outcomes (e.g., benefits), so guidance on how the different goals should be elicited, defined and organised is not provided. As such, it is difficult to know which goals are being examined for alignment to expected benefit, and consequently when to stop abstracting goals (i.e., the graph’s boundary).

| Table 4.15: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported) |
|---|---|---|---|---|---|---|---|---|---|
| ✔ | ✔ | ✘ | ✔ | ✔ | ✔ | ✘ | ✔ | ✔ | ✔ |

**Summary of Approach #14: GRL + (Key Performance) Indicators**

**Summary:** Uses GRL for Business Intelligence (BI) by allowing real world numbers (KPIs) from BI data warehouses to be mapped to GRL’s subjective-quantitative contribution scores, for monitoring the historic ‘health’ of the goals.

**Pros & Cons:** Allows goal contribution to be specified refutably (reducing ambiguity), but is ignorant of uncertainty, non-linear goal contribution, and quantitative propagation of partial goal satisfaction to higher level benefits.

### 4.5.9 GRL + Balanced Scorecard (BSC) + SWOT

Kaplan & Norton’s Balanced Scorecard (BSC) & Strategy Maps approaches [KaNo96a] offer guidance on formulating business goals and related measures under four perspectives (purportedly providing complete coverage of business concerns): Financial, Customer, Internal Processes, Learning & Growth. BSC was intended to guide organisations away from monitoring purely financial goals, which tend to be lagging rather than leading indicators\(^\text{25}\), hence supporting proactive rather

---

\(^{25}\) E.g., by the time ‘profit’ (lagging) is accounted for, it is too late to influence it. However, if ‘customer complaints’ (leading) is monitored, action can be taken to improve profit [Pers13, p.74]. (Pro- vs. Re-active).
than purely reactive decision making. As such, BSC could be useful for aiding elicitation and completeness analysis of the goals representing a software capability’s business benefits.

A Balanced Scorecard is ordered hierarchically, where goals feeding up from the bottom (internal enterprise aspects such as ‘Learning and Growth’) are the outcome of strategic planning from the top (external enterprise aspects such as ‘Financial’). To visualise the cause-effect goal hierarchy, Kaplan & Norton propose the ‘Strategy Map’ diagram, which allows the value and role of intangible goals at the bottom of the hierarchy to be understood, thereby showing their ‘Strategic Alignment’ [GiPZ11]. Case studies in industry indicate that this is its most useful feature: The practitioners appreciated the assistance in the definition and communication of strategy, as well as for the assistance in adjusting the strategies according to the verification of assumptions on the adequacy of lower level goals for higher goals [Ahn01, p.457]. (This is highly similar to the use of goal graphs in RE.)

Balanced Scorecards are reportedly used by most Fortune 1000 companies [FuJe11], which by itself is not a reliable indication of its usefulness (indeed, many practitioners claim it is implemented because it is fashionable to do so [Prei12], and so did so without management buy in). However, it does indicate that the notion of modelling basic cause and effect relationships will not be completely alien to, and hence unusable by, business practitioners. However, BSC’s causal links between goals are qualitative and un-weighted, which often lead to them being contradictory, non-descriptive, trivial, or inefficient. (Indeed, studies of the balanced scorecard show that the causal chains not defined by decomposed numerical variables can be less mechanistic (e.g., $A \rightarrow C$ rather than $A \rightarrow B; B \rightarrow C$) and descriptive, often requiring further “arguments in favour of links” [Ahn01, p.453]. BSC’s unguided modelling of causal links also often leads to ‘overwhelming’ complexity [Ahn01, p.453], since when non-numerical and hence imprecise goals are defined, they can be vaguely interpreted, and hence have far more plausible relationships to other goals than they would if they were precise).

Bringing the discussion back to RE, numerous researchers have proposed integrations between BSC’s four goal dimensions and GORE languages, thereby allowing a requirements model to span business as well as software concerns. Babar et al. propose the SMi* approach, adding actors and dependencies to Strategy Maps using i* [BaZC10]. Similarly, Barone et al. propose an approach for business strategy modelling using GRL goal graphs, where goals are visually tagged with their membership to one of BSC’s four perspectives [BJAM11]. Additionally, Barone et al. propose the addition of a new ‘Situation’ element to GRL for modelling opportunities and weaknesses (i.e., ‘W’ & ‘O’ from ‘SWOT’ [Iiba09]), e.g., ‘High customer complaints’ is a weakness for goal ‘More products sold’. However, these elements may be an unnecessary inflation of the comprehensive GORE languages, E.g., a ‘Weakness’ could simply be an ‘Obstacle’ [LaLe00], and an ‘Opportunity’ a goal to be maximised.
4.5.10 GRL + e3 Value Modelling

Gordijn et al.'s e3 value modelling framework (based on Porter’s popular Value Chain Analysis method [Port98]) has recently been integrated with i* (and hence GRL) into a requirements modelling process for e-service design [GoYR06]. Essentially, e3’s value-exchange links between e3 actors (providers and consumers) are mapped to i* resource dependencies between i* actors (dependers and dependees). For example, in an online radio business: the ‘Musician’ actor exchanges ‘Right to make music public’ with the ‘Rights Society’ actor in return for a ‘Fee’. However, it is not made explicit how nor why value exchanges generate value for their actors. This is especially pertinent for value exchanges not involving financial transactions, e.g., where the ‘Listener’ provides an ‘Audience’ in return for a ‘Radio Stream’. Instead, the e3 approach requires the modeller to provide a direct financial point estimate ($) for each value exchange (as is similar to Chen’s approach, later discussed in 4.5.13), and guides the modeller to use financial accounting analysis techniques such as discounted cash flow [LaVo12] to do so. However, neither the chain of instrumental value generation behind such valuations (in this context, leading to a source of financial intrinsic value), nor uncertainty, nor assumptions (e.g., about dependent variables or necessary actions) are made explicit. As such, an e3 value model is more of a rationale-less externalisation of the modeller’s belief, than a reasoned, robust, and refutable argument for a requirement’s value.

Finally, it is pertinent to mention the wider field of ‘business modelling’ from which e3 draws its concepts, whose purpose is to “describe the transfer of economic value between involved actors—economically independent entities” [ZdSG13] (derived from [OsPi10a]). Resulting from Osterwalder’s business model ontology, itself resulting from a review of other such ontologies [Oste04, p.45], is the popular ‘Business Model Canvas’ [OsPi10b]. These tools have obvious potential for input into the value analysis of software requirements, since they encourage the early-elicitation of 9 ‘building blocks’ consisting of partners, customers, value propositions, resources, and so on. Gordijn et al.'s
approach discussed in this section simply exploits the conceptual overlap between the ‘building blocks’ of business modelling and the language constructs of goal modelling in RE. Finally, Osterwalder’s ontology has relationships between value propositions and the capabilities required for them [Oste04, p.106], which would enable rudimentary ‘software capability→value’ causal propositions, (i.e., only of the ‘necessary’ causation type and of qualitative nature).

4.5.11 GRL + Consumer Values

Zdravkovic et al. establish traceability between system requirements and Holbrook’s consumer values (which drive human affection for products — Chapter 2.1.3) [Holb98] using i* (hence GRL) [ZdSG13]. The aim of their approach is to guide feature selection in Software Product Lines (SPL) [PoBL05] by the consumer values that the software’s end-user(s) are driven by. However, such traceability could also be useful for guiding the elicitation of a requirement’s ‘internal’ benefits to an end-user. (Consumer values are most relevant to end-user stakeholders since they directly ‘consume’ the software, but non-end-users could also be, for example, ‘Stimulated’—Holbrook’s ‘Play’ consumer value.)

Zdravkovic et al. propose to model each consumer value using a GRL belief element owned by an actor, and a respective GRL softgoal prescribing the maximisation of the consumer value. For example in the context of an online education software system, a student’s softgoal ‘Ethics be Satisfied’ (Holbrook’s ‘Ethics’ consumer value) is refined to the system’s soft goal ‘Cheating be Prevented’, which is refined to the hard goal ‘Online Behaviour be Logged’ [ZdSG13, p.13]. Hence, the software’s behaviour logging capability is modelled as a means to the end, where the end is the satisfaction of the student’s ‘Ethics’ consumer value. In addition to such top-down refinement of requirements from consumer values, bottom-up goal abstraction is also possible. For example, the benefit of a healthcare management system’s requirement to ‘Give nurses writing access to patient records’ for a Doctor could be explained through the ‘Status’ and ‘Excellence’ consumer values [PPWW10], which infer a negative attitude toward manual data entry. Such qualitative benefits might be either the software’s primary expected benefits, or their desirable ‘side-effects’, depending on the viewpoint of the evaluator. (A
similar but less grounded-by-theory guideline is Cockburn’s proposal that software requirements should describe the “SINs” (Styles, Interests, and Needs) of the stakeholders [Cock10], where the ‘Interests’ concept is not too conceptually distant from Holbrook’s consumer values.

Consumer values play an important role in explaining a stakeholder’s motivation for even the most tangible of expected benefits. However, Zdravkovic et al.’s integration of goal graphs and consumer values is neither fully efficient nor sufficient for this thesis’ problem. Aside from the capabilities that Zdravkovic et al. did not attempt to address shown in Table 4.18, other areas could be improved. Firstly, it is likely that most stakeholders would desire every consumer value to some extent, but that some extent is not captured or represented in the GRL model. Indeed, a stakeholder’s desire for something (including consumer values) is often a value judgement over two or more competing values [Bear65, p.16], e.g., a preference for ‘Ethics’ over ‘Play’ (excitement). Secondly, consumer values and their satisfaction (and hence a requirement’s contribution toward them) are treated qualitatively, whereas quantitative reasoning about them is possible and less ambiguous. As an additional concern, requiring two GRL elements for each desired consumer value (potentially all of them) unnecessarily bloats goal models, which can already grow unmanageably large [BlCV06, MoHM10].

Table 4.18: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>X</td>
<td>X</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>

**Summary of Approach #17: GRL + Consumer Values**

**Summary:** Establishes traceability between consumer values (things that motivate human affection for products) and software requirements in i* models. Intended to aide selection of features in software product line engineering.

**Pros & Cons:** Explains the psychological and ‘personal’ benefits for software capabilities. However, the approach does not attempt to define scales of measure for Consumer Values, nor degrees of contribution toward them.

### 4.5.12 GRL + Stakeholder Agreement

Hassine and Amyot propose a framework for validating GRL models [HaAm13] using automated generation, distribution, and statistical analysis of online questionnaires from the GRL models. For example, for each contribution link, the attitudinal question “The execution of <Task-1> <Helps> the realization of <Goal-1>; please tell us to what extent you agree with this statement? {5 point Likert scale from ‘Strongly agree’ to ‘Strongly disagree’}” is presented to the stakeholders. Hence, the degree of stakeholder agreement about the modelled hypotheses can be ascertained, which, in the context of this thesis and a GRL contribution link, would be that a software requirement will cause some degree of benefit.
While this approach supports the communication of the goal models amongst stakeholders (which is necessary for it to be beneficial to the RE process [PiKa11, LiDF11]), it remains unclear how scalable the approach would be for real world sized projects, or how to select the questions given constraints on the stakeholders’ time. Furthermore, the approach assumes that stakeholders will understand the semantics of GRL’s elements and links. As such, differences in opinion about extents of goal contribution are not likely be captured, since for example, to the untrained stakeholder, GRL’s ‘Help’ label would simply imply a positive contribution (which is far-less refutable compared to its more precise meaning discussed in Section 4.5.2, or compared to correlation as discussed in 4.4.2).

Table 4.19: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✅</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✅</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✅</td>
<td>✅</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>

Summary of Approach #18: GRL + Stakeholder Agreement

Summary: Adds to GRL the capability to automatically generate online surveys polling stakeholders on their agreement with the modelled knowledge, e.g., that a requirement (task) will satisfy some stakeholder desire (goal).

Pros & Cons: Lack of stakeholder interaction is a serious RE defect, and asking stakeholders to evaluate the correctness of a goal models is an intuitive means to that end. However, its scalability and practicality is unclear.

4.5.13 Goal Graphs + Earned Business Value Annotation

Chen et al. annotate generic goal graphs (i.e., not specifically using GRL, KAOS, i*, etc.) with preferences over software features (behavioural goals) in their approach for self-adaptive software systems [CPYZ11]. Chen et al.’s aim is that under heavy load, a website system should be able to disable features to provide adequate resources to the features that earn the most business value, rather than bringing the whole system down (or allowing less business critical features to run), as motivated by incidents such as Amazon.com’s 2010 three hour downtime, costing $51,400 per minute. Hence, their approach rests on the assumption that each functional ‘factor’ (groups of behavioural goals in a goal graph, or “a kind of runtime system transaction”) can be assigned a financial gain per use. For example, they model that if a customer posts a review of a product, $0.032 of value is earned on average through positive social advertising, whereas each product page view is modelled to earn $0.08 on average (calculated via the conversion rate of page views per purchases).

Unfortunately, explanations of those calculations, nor simple guidelines for eliciting such financial estimates are not provided: “how to elicit relative parent values can be induced as an economic problem of consumer behaviour analysis, which can and shall be resolved by economics experts” [CPYZ11], — who are not likely to be readily available in a typical software development project due to expense or privacy. Conveniently for the paper, the online shopping system’s goal graph is
decomposed from a root goal that entails a financial transaction, and whose subgoals have easily countable frequencies-of-satisfaction, thanks to the web server’s built-in logging functionality. In other words, satisfaction of the ‘online shopping’ goal (when the customer makes a purchase) directly increases the business’ profits by some calculate-able ‘average $-per-goal-satisfaction’ amount, and the software capabilities used to create that sale can be attributed with a measure of importance based on their frequencies of use. However, this approach’s requirement for financial value estimation of root goals is less applicable for systems whose ‘root’ goals do not entail direct financial transactions, which is the case for the majority of software in non-commercial contexts.

To arrive at value estimates for behavioural subgoals (e.g., ‘Checkout’ is a subgoal of ‘Online Shopping’), Chen et al.’s approach elicits ‘relative parent values’ using the question: “If the hard goal is unbound, what is the probability of not achieving its parent goal when all other sibling subgoals have been achieved?”. E.g., ‘Checkout’ → ‘Online Shopping’ is rated as 1, while ‘Recommend Products’ → ‘View Product Details’ is rated as 0.1. Sparse guidance is offered for eliciting these ‘degree of necessity’ values, other than to ask the above question. (Opportunities for improvement include that estimates could be ascertained by conducting ‘AB’ tests [SiKo13] or aided by inspecting server logs.) More importantly, these seemingly logically-derived probability values are ambiguous for describing the value of a capability, due to ambiguity in their timeframe, among other reasons. For example, if the ‘Checkout’ capability becomes offline after a shopper has selected the items, it is not true that in 100% of cases (as implied by the ‘1’ probability) the basket of goods will not be purchased, since some shoppers will retry later. Furthermore, the mechanism by which a subgoal is to some degree necessary for a parent goal, and hence the extent to which it is valuable is ambiguous. For example, the ‘Make review’ → ‘Online Shopping’ link is rated as 0.1, yet it is not clear how it leads to completion of the same ‘Online Shopping’ experience that the ‘Checkout’ process was previously considered to be essential for. It might be argued to belong to a longer causal chain whereby ‘Make review’ is necessary for the existence of product reviews that contribute to sales.

However, this leads to inconsistency since ‘Make review’’s 0.1 rating is in the context of many-shopping-sessions, but ‘Checkout’’s 1 rating is in the context of one-shopping-session.. In short, far greater precision is required than is afforded by Chen et al.’s approach to be able to rationally calculate the value of a subgoal based on its probabilistic degree of necessity to a parent goal whose value is known, (or in other words, to answer ‘what is the cost/benefit of the subgoal?’).

Finally, non-linearity and conceptual precision is not considered by the approach, since, for example, every product review is modelled to create the same value, whereas a negative product review is likely to cause financial loss (where there are no substitutable products), and the first positive review likely creates more value than the hundredth. Additionally, Chen et al. base their causal predictions of value generation (future behaviour) entirely on correlation (past behaviour), e.g., that increasing the number of page views will increase sales just as it did in the past. This is not true for most causal relationships [Ster00, p.141], and so some element of ‘belief’ and description of uncertainty about the future seems sensible, especially when making economic predictions such as for the ‘mean profit per transaction’ goal. Overall, since Chen et al.’s approach is designed for machine use rather than human use, the intention is not to provide an argument for how or why value is generated by software capabilities, but rather to declare which capabilities to prioritise in a self-adaptive scenario of a web service to survive under load. However, even in this context, the approach seems suboptimal since the root cause of the problem-causing overload is not considered, and hence some software capabilities are likely to be disabled for the result of zero positive effect on the load.
It is pertinent to note that numerous researchers have applied financial investment portfolio optimisation techniques, such as the Capital Asset Pricing Model, to requirements selection [SiNu01]. However, eliciting the required ‘expected return’ (value) and ‘risk’ (probability) variables for each requirement, proved difficult “due to insufficient relevant data” [SiNu01, p.4]. Versendaal et al. treat the problem of requirements selection using Integer Linear Programming (ILP) [VVBD05], requiring estimates on variables such as a requirement’s cost, revenue, deadline-impact, etc. However, the reviewers of Versendaal’s work stressed the approach’s “vulnerability to the quality of the [input] data”, since ‘the estimation of a requirement’s revenue-earning potential’ “is enormously hard” [KaGS05, p.17]. The final noteworthy financial value estimation technique applicable to requirements valuation is the direct elicitation of a stakeholder’s ‘Willingness to Pay’ [Cook03], i.e., the maximum financial amount that a stakeholder would pay in order to have the requirement satisfied. While this technique has not been thoroughly researched for RE, its well-established vulnerability to cognitive biases (especially ‘scope neglect’, ‘part-whole bias’ [BaGr96] and ‘budget constraint bias’ [MiCa89]), support the argument that eliciting financial valuations of requirements directly from stakeholders would not be robust.

Table 4.20: Feature Comparison Table (Key: ✔️ Explicitly, ✅ Somewhat, ✘ Not} Supported)

|------------------------------|---------------------------------|---------------------------|--------------------------------------|--------------------------------------|---------------------------|----------------------|-------------------|-------------------|------------------------|-----------------|-----------------------------|-----------------------------|--------------------------|------------------|------------------------|---------------------------|-----------------|------------------------|----------------|
The framework draws on Loucopoulos’ S3 framework [Louc03] to illustrate the use of system dynamics in goal modelling. A case study of an RE project for the Athens 2004 Olympic Games’ venue planning is described, whereby the stakeholders found it “relatively easy” to identify distinct functional and quality aspects of the system, but found it difficult to quantify them, (as is common in RE [Gilb07]).

The case study introduces the example goal to “Minimise the time that a customer has to wait in order to get serviced” [Louc03, p.9] for the services provided at the stadium, which dictated, for example, the number of hotdog stands to be set up. The lower the target for the goal, the more resources required to achieve it, and since the stakeholders could reach agreement, Loucopoulos and Preskas constructed system dynamics models to understand the effects of, for example, the number of hotdog stands on the average customer waiting time. Firstly, goal graph refinements were derived according to supply and demand variables, e.g., for the waiting time goal the variables were:

- **supply** (subject of decision): number of hot dog stands, and their service rate per minute;
- **demand** (assumptions, not decisions): number of hot dog customers per minute.

Secondly, plausible scenarios were proposed for each of these variables, e.g., 15 hot dog stands, 3 customers served per minute, etc. Then, these supply and demand variables were simulated over increments of the chosen unit of time (e.g., throughout the minutes of a day), resulting in a hypothetical mean and maximum values for the ‘waiting time’ goal. Hence, the effects of various degrees of possible satisfaction for the planned cause (referred to as a ‘design choice’, e.g., the number of hotdog stands), could be investigated, thereby allowing an exploration of the ‘dose-response’ relationship for various ‘doses’, where each dose corresponds to one ‘design choice’.

However, no guidance on modelling the resulting dose-response relationships is provided, nor their epistemic and aleatoric uncertainty and non-linearity, therefore only supplementing the construction of goal models rather than integrating with them. Furthermore, the intention of the approach is not to model the instrumental value of ‘design choices’ (e.g., ‘number of hotdog stands’), nor is it to model the rationale for choosing among alternative goals. Hence, the concepts of goal ‘contribution’ and contribution propagation (through chains of means-ends) do not exist in the approach, instead treating reductions in ‘customer waiting time’, for example, as intrinsically good. Indeed, the authors wrote “Should we be happy with 30 seconds waiting time or with 15 minutes? In some cases even that answer was not ready, so it had to be negotiated” [LoPr03], indicating that neither the rationale for ranges of waiting time, nor the consequences of ‘good’ or ‘bad’ waiting time, were explored or modelled. Hence, the resulting argumentation for the chosen goal quantifications (e.g., the number of hotdog stands) is neither easily refutable nor explicit, since the various stakeholders reading the results of the simulation might have different beliefs about acceptable waiting times.

(More recently, Nalchigar et al. proposed guidelines for mapping entities between goal graphs and causal loop diagrams [NaYE14], but due to the considerable conceptual overlap with Loucopoulos and Preskas’s earlier work (without citation), they are considered equal in the feature comparison below.)
4.6 IT Alignment Approaches

Researchers have proposed several approaches to validate the alignment between technological strategies (means) and business strategies (ends), in order to maximise competitive advantage.

4.6.1 3g Alignment Framework

Singh and Woo propose the ‘3g’ framework, building on i* strategic rationale diagrams to explore the business-IT alignment of software requirements. Their work is motivated by the observations that “strategic alignment research is limited and devoid within the requirements engineering discipline”, especially on “distinguishing between assigned and interpreted goals” [SiWo09], where:

- an **interpreted goal** is a stakeholder’s interpretation of a goal implicitly or explicitly assigned to them, where the interpretation “may be subjected to the agent’s personal traits such as their experience, motivation and their perceived conception of the task”;
- an **assigned goal** is an operational level goal, defined by executives and middle managers.

Singh and Woo justify this delineation with an example: A sales clerk may be assigned the goal “to process customers’ transactions in a timeliness [sic] and seamless manner to ensure consumer satisfaction”. However, the clerk’s opinion (interpretation) of their goals could either be in a “Conflicting”, “Satisficing”, or “Better” state of alignment, ordered by desirability. Only the ‘Satisficing’ state is in alignment, since the other two states require a change in either the interpreted goal or the assigned goal, respectively. For example, when the clerk is asked to describe the goals of their tasks, they might respond with “completion of customers’ transactions”, indicating a ‘Conflicting’ state of alignment. Alternatively, if the clerk replied “[to] process customers’ transaction efficiently and elicit feedback from the customer during the transaction period in order to ensure consumer satisfaction”, then a ‘Better’ state of alignment is indicated [SiWo09].
Crucially, Singh and Woo claim that “vastly” different sets of software functionality would be specified for the intended system, depending on the state of goal alignment. For example, in the case of ‘Better’ alignment, a point of sale system might be specified with “an additional feature for the sales clerk to record customers’ suggestions”, while in the case of ‘Conflicting’ alignment, the system might be specified to only fulfil basic, non-differentiating operations. However, this whole argument rests on the assumption that the requirements engineers would either consult only the sales clerks for the system requirements, or that the managers who defined the assigned goals are themselves incapable of proposing system features. Regardless, it is well known that consulting a wider range of stakeholders when eliciting requirements will lead to a wider set of perspectives, and hence more innovative and therefore valuable system functionalities. In other words, there is an “inadequate stakeholder input” RE defect [LiFi12]. Hence, the main ‘take away’ point of Singh & Woo’s framework could be summarised as the guidance to consult those responsible for setting strategic goals, as well as those responsible for their fulfilment, about how they would like the goals to be implemented by the system-to-be, and then both groups’ ideas should be shared for feedback.

Finally, Singh & Woo provide two generic questionnaires: one for eliciting the strategic goals of an organisation (e.g., “What are the opportunities and threats in the industry?”, and “What is the firm’s current short-, intermediate- and long-term strategy?”), and another for eliciting the stakeholder’s interpreted goals (e.g., “What purpose does the performance of your task serve in the firm?”) [SiWo09]. Answers to the first set of questions would undoubtedly be useful for deriving expected benefits, but the questionnaire assumes that the answers are known by the stakeholders and are merely undocumented, whereas the reality is that many stakeholders will be unable to provide the answers due to lack of knowledge, or the confidentiality of competitive strategy. Furthermore, Singh & Woo’s framework was applied in an academic environment without industrial involvement or empirical evaluation, and no tool support exists. As such, the usefulness of the framework’s novelties relative to their usability remains unclear. Nevertheless, recent work on assessing stakeholder goal congruence [MiMa12] and agreement in i* models [HaAm13] has shown to be promising.

Table 4.22: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>

Summary of Approach #21: 3g Alignment Framework

**Summary:** Explores conceptual (rather than causal) alignment between proposed software requirements, the actions that the stakeholders think they should perform, and the strategic goals that should be realised by actions.

**Pros & Cons:** The approach visualises ‘line of sight’ from the tasks stakeholders perform with the system, to the benefit realised for the organisation. However, modelling is entirely ‘soft’ and qualitative, and so can be ambiguous.
4.6.2 B-SCP (i* + OMG’s Business Motivation Model)

Bleistein et al.’s Business Strategy, Context, and Process (B-SCP) framework [BCVP06] for validating the strategic alignment of IT requirements combines i* goal modelling and the OMG’s Business Motivation Model (BMM)’s [Obj10] concepts for the strategy theme, Jackson context diagrams [Jack00] for the context theme, and Role Activity Diagrams [Ould95] for the process theme. The mapping between the i* language and the six BMM concepts ("Goal types" and "Activity types" in Table 4.23) is of most interest to this thesis, since it concerns means-ends modelling.

<table>
<thead>
<tr>
<th>Goal types</th>
<th>RE equivalent</th>
<th>Definition</th>
<th>Activity types</th>
<th>RE equivalent</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision</td>
<td>Soft goal</td>
<td>An end-state toward which the organization strives</td>
<td>Mission</td>
<td>Task</td>
<td>The primary activity of the organization that achieves the vision</td>
</tr>
<tr>
<td>Goal</td>
<td>Soft goal</td>
<td>An abstract statement of intent whose achievement supports the vision</td>
<td>Strategy</td>
<td>Task</td>
<td>A long-term activity designed to achieve a goal</td>
</tr>
<tr>
<td>Objective</td>
<td>Hard goal</td>
<td>A specific and measurable statement of intent whose achievement supports a goal</td>
<td>Tactic</td>
<td>Task</td>
<td>A short-term action design to achieve an objective</td>
</tr>
</tbody>
</table>

For example, Figure 4.12 highlights the path from leaf-level tactics (representing requirements on an IT system) toward longer-term business objectives, and ultimately more abstract business goals.

![B-SCP Diagram: Strategic Alignment of IT Requirements](image)

Figure 4.12: B-SCP Diagram: Strategic Alignment of IT Requirements (modified from [BCVP06]). Key: ‘O’ = business objectives, ‘T’ = tactics, (and the leaf level tactics = IT system requirements).
Figure 4.12 establishes traceability from requirements to goals defined at a business’ operational, tactical, and strategic levels, thereby complying with Aurum and Wohlin’s recommendations for value-based RE [AuWo05]. However, B-SCP’s objectives are not quantitatively defined (despite the BMM’s stipulation), and contribution strengths are not attached to links between requirements and objectives. Hence, B-SCP does not model the extent to which a requirement is believed to satisfy an objective/goal (and hence the extent to which the requirement complies with the business strategy). For example, the alignment of software requirements to an objective to reduce costs depends on the extent of reduction that would be contributed. Moreover, B-SCP’s methodology refines business strategy top-down towards IT requirements, which means that completeness of the model is dependent on the completeness of the business strategy. Additionally, B-SCP proposes a simple version of i* with a reduced set of modelling elements, and omitting richer GORE concepts such as AND/OR refinement, obstacles, and actors, that would enable the modelling of decisions amongst alternatives (and hence the baseline for value estimates), barriers to value realisation, and the stakeholders who stand to gain or lose from a software capability’s implementation.

B-SCP replaces i*’s actor elements and dependencies with ‘domains of interest’ and ‘shared phenomena’ from Jackson context diagrams (mapped to roles in RADs), so as to primarily be able to model “how organizational entities relate with each other independently of organizational goals, tasks, and other requirements” [BCVP06, p.864]. However, i* was designed with the premise that organisational structure and dependencies should be “organised around outcomes [i.e., the satisfaction of goals]”, and those that are not should be candidates for business process re-engineering [Yu95, p.71]. (For completeness, i* and GRL do have the capability to model entity relations ‘independently of organizational goals’, e.g., that an actor can be “part of a larger whole” [Yu95, p.17].) While there are advantages to be gained by the use of Jackson’s context diagrams, this thesis considers i*’s actor metamodel sufficient for modelling the stakeholders and software components involved in chains of instrumental value derivation, and requiring stakeholders to learn another modelling language would further obscure the framework’s usability, especially in organisations using only Volere et al.

Goals and processes are naturally related, given that the achievement of a goal should be a process output [Ould95]. Consequently, numerous approaches integrate process models with goal models, e.g., with Business Process Modelling Notation (BPMN) [VaSP13], UML’s Sequence Diagrams [GhKC07], or URN’s Use Case Maps (Section 4.5.2). Liaskos et al. even propose to model sequence within a goal model, annotating goal decompositions with their ‘precedence order’ [LMSM11]. B-SCP’s adoption of Role Activity Diagrams (RADs) enables linkage between business processes and a goal model’s goals without further complicating the goal model, and in a visual notation argued to be more flexible and easily understood by non-trained stakeholders [FPKJ11, p.421] than other notations and standards such as BPMN [Omg08]. (Though some business process modelling best practice argues that BPMN and RAD offer complementary perspectives [Mier06, p.42]). This thesis’ focus is more on understanding the value of tasks performed by software agents, rather than on the sequence of those tasks, and so goal models are considered sufficient for that purpose. Nevertheless, a holistic software engineering framework should certainly consider the interaction and traceability between processes and the value created by the sequenced interaction of the system’s actors.

Owing to the BMM’s standardised language for describing business strategy concepts, numerous researchers have proposed the relation of software requirements (using non-standard or standard languages such as SysML [SoVr08]) to the BMM’s six elements in order to show strategic alignment [Amsd08, Berk06, BrZa08, CuPa12, LiPC08]. However, as is visible in Figure 4.12, the ‘Tactic’ (T) (action) and the ‘Objective’ (O) (desired result) BMM entities are the most used of the six. Indeed,
whether or not stakeholders would understand the differences between the six elements described in Table 4.23 (and whether it is worth distinguishing between them) remains unclear. This researcher’s experience with software and business practitioners has found frequent confusion about the distinction between the different terms. Furthermore, there is an assumption that stakeholders would be able to articulate these BMM entities representing their organisation’s strategy (after ‘VMOST’ (Vision, Mission, Objectives, Strategy, Tactic) analysis [Soud99]), yet it is claimed in the business literature that “a mere 7% of employees today fully understand their company’s business strategies and what’s expected of them in order to help achieve company goals” [KaNo01].

Sommerville et al. point out that many organisations may not even have a set of business objectives or goals [SoLS12, p.14]. Indeed, a survey of corporate managers in Europe concluded that only half of the respondents’ organisation’s strategies were documented [BlHo05, p.70], and even if a documented strategies do exist, it is likely to be accessible only by senior management due to commercial sensitivity. This affects B-SCP, given that it refines business strategy down toward IT requirements [BCVP06, p.850], which means that completeness of the model (and usefulness of the approach) is dependent on existence and completeness of the business strategy. Moreover, B-SCP offers no guidance for abstracting software functionality upwards, in order to propose new strategy or expected benefits, (hence restricting the role of the software designers to ‘fulfillers of business strategy’ rather than contributors to its definition).

Finally, Veres et al. propose a machine readable ontology for B-SCP (in the RDF language) to ease the comprehensibility of large models that are unavoidable in practice, and to enable general purpose knowledge modelling software tools to support the approach [VSBC09]. However, the efficacy of goal modelling in such tools compared to purpose-built tools has not been studied.

It is pertinent to mention three IT strategy alignment approaches similar to B-SCP: ARMOR [EnWi12], INSTAL [ThSa07], and ServAlign [LaBA10]. ARMOR extends the Enterprise Architecture ‘Archimate’ modelling language with GORE concepts, permitting traceability between ‘Architecture Components’, ‘System Requirements’, and ‘Business Goals’ using qualitative goal contribution links. However, Enterprise Architecture frameworks (TOGAF, Zachman, etc.) pursue high level relations between Information Systems (IS) and organisational objectives. They do not provide support for lower-level concepts (e.g., system requirements) [EnWi12], nor do they address states of misalignment [Akhi14]. INSTAL similarly uses qualitative contribution links for establishing traceability between components of ‘[Business] Strategy Documentation’ and ‘Information System Documentation’, while ServAlign uses i* to qualitatively model traceability between ‘Functional Goals’, ‘Business Plans’, and ‘Maximise/Minimise’ goals. Due to the considerable conceptual overlap between these three approaches and due to space constraints, B-SCP is considered as best-case representative of this theme of approach in Table 4.24’s feature analysis table.
### 4.6 IT Alignment Approaches

#### Table 4.24: Feature Comparison Table (Key: ✓ Explicitly, ✅ Somewhat, ✘ Not) Supported

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Summary of Approach #22: B-SCP (i* + OMG’s Business Motivation Model)**

**Summary:** Could be used to guide the elicitation of the benefits of software requirements to a business using traceability to the business strategy concepts standardised in the OMG’s Business Motivation Model (BMM).

**Pros & Cons:** Translates operational goals (i.e., satisfied by the system in seconds) through to strategic goals (i.e., satisfied by the business in months/years) to explain how the software requirements will create advantage to the business. However, contribution links between requirements and goals are not quantified, and so assumes full sufficiency.

### 4.6.3 Balanced Scorecard (BSC) + Multiple Linear Regression (MLR)

Fukushima and Peirce propose a business decision making framework based on the Balanced Scorecard (reviewed in 4.5.9) and multiple linear regression [FuJe11]. Their aim is to enable decisions about IT (which could conceivably be software requirements decisions) to be made in the context of higher level business objectives, in a less subjective manner than is otherwise the norm when setting balanced scorecard weights [ItLM03]. Fukushima and Peirce intend for their framework to remedy two problems [FuJe11, p.4]:

- “[An] organization cannot evaluate how improvements in its internal business processes will contribute to a bottom line”. Applied to this thesis’ context: what is the contribution’s extent?
- “[An] organization would focus on inappropriate objectives” due to misunderstanding the ‘actual’ causal relationships. Applied to this thesis’ context: are the contributions refutable and re-used?

Essentially, hierarchical and orthogonal causal relations between balanced scorecard goals are represented as equations comprised of explanatory variables, as exemplified by Figure 4.13.

![Figure 4.13: Describing Balanced Scorecard vertical goal relationships with a regression model [FuJe11]](image)

---

129 | 4.6 IT Alignment Approaches
Given such a regression model, a Monte Carlo simulation can suggest a probability for achieving a high level goal to a certain degree, given a change to a low level goal, (e.g., a 30% increase in software usability might increase the software’s Net Present Value by $50m with a probability of 0.36).

While such a capability seems powerful and in line with the capabilities desired by this thesis’s framework (if the BSC goal hierarchy is extended to software capabilities), there are challenges:

- Primitive causal proposition dimensions (e.g., necessity & sufficiency), nonlinear weighting, and actors are not considered, meaning that the model is vague about the whys, how’s, and who’s. E.g., ‘Installability’ (Figure 4.13) (machine domain) is not sufficient for ‘Customer Satisfaction’ (application domain), but it is likely to be necessary for a causal component of it.
- Historic data from a similar context to the current software project is required to build and apply such regression models. That Fukushima and Peirce’s 2011 paper [FuJe11] uses coefficients (Figure 4.13) from a 1995 IBM customer survey [KeKS95, p.4] shows that such data is rarely available. While this problem is inherent to this thesis’ problem, it is compounded in this approach since epistemic uncertainty about the model or the parameters is not considered.
- The goals’ scales are mostly context independent, aggregated, non-mechanistic, and refer to subjective Likert scores (e.g., Figure 4.13’s ‘Reliability’ explanatory variable is an arbitrary ordinal scale of IBM’s customer satisfaction with their software products’ reliability). Such proxy indicators cannot be directly modified by the software designers, and so do not describe what will actually be changed in the project’s particular application domain [Jack95a].

Table 4.25: Feature Comparison Table (Key: {✔️ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

**Summary of Approach #23: Balanced Scorecard (BSC) + Multiple Linear Regression (MLR)**

**Summary:** Could be used to predict the value of software requirements once a sizeable database of past experience has been built. Value oriented decisions could then be aided by Monte Carlo simulations to assess aleatoric uncertainty.

**Pros & Cons:** Coefficients describing contributions are flexible, but are simplistic (nonlinear), and dimensionless-ness encourages scale ambiguity. Epistemic uncertainty is not considered, so believability of the approach’s output is an issue. Trade-offs between conflicting requirements (i.e., weights) could be guided using its cited regression model.

### 4.7 Benefits Realisation Approaches

Numerous researchers have proposed methods for planning and modelling the realisation of benefits from IT interventions, due to the ever-increasing complexity of realising their value [SaHK08, p.61].
4.7.1 Thorpe’s Results Chain (as in Boehm’s Value Based Software Engineering)

The ‘Benefits Realisation Approach’ is a principle component of Boehm’s original Value Based Software Engineering proposal [Boeh05], and is operationalised by Thorpe’s Results Chain [Thor99] which visually demonstrates traceability between an initiative (i.e., a new software system, rather than individual requirements) and its outcomes (i.e., benefits). The visualisation is a directed acyclic graph, where nodes represent ‘Initiatives’ (squares/rectangles), ‘Outcomes’ (circles/ovals), and ‘Assumptions’, while edges (arrows) represent ‘Contribution’ links, as in Figure 4.14. Hence, a Results Chain could be considered a simplified goal graph, except for that it only shows cause and effect, and does not include GORE’s concepts such as AND/OR refinement (decomposition), agents, beliefs, etc.

A Results Chain’s contribution links allow the modelling of one initiative to many outcomes, but the links are not defined quantitatively. For example, outcome “reduced time to deliver product” can contribute to outcome “increased sales”, if assumption “delivery time is an important buying criterion” holds true [Boeh06, p.2] — but the ‘dose-response’ relationship between “delivery time” and “sales increase” is not explored, even for just one ‘dose’ such as a ‘50% delivery time reduction’. This is problematic when outcomes are business objectives (which are quantitative in nature [Obje10]), since their satisfaction, and hence the sufficiency of the initiatives, depends on the extent that they are contributed to. However, Thorpe does propose that outcomes should ultimately be defined measurably, in order to assign ‘executive accountability’ for their realisation, and to enable retrospective evaluation, i.e., after a system has been implemented and deployed [Thor99, p.183].

![Figure 4.14: Results Chain (squares represent initiatives, and circles represent outcomes)](BoHu03)

Numerous similar approaches for modelling the expected benefits of an initiative exist under the ‘Benefits Realisation Management’ umbrella [SaHK08] (such as Peppard & Ward’s ‘Benefits Dependency Networks’ [PeWD07]), but in the context of this thesis, this family of approaches are considered, in essence, to be equal to Thorpe’s. Indeed, other approaches may actually obfuscate the key concerns. For example, Benefits Dependency Networks distinguish between “Means (e.g., ‘IT Enablers’), “Ways (e.g., ‘Business Changes’), and “Ends (e.g., ‘Benefits’)” [PeWD07], but this distinction is purely contextual (what is a ‘Way’ in one context is an ‘End’ in another), and has significant conceptual overlap (‘Ways’ lead to ‘Ends’, and so are highly similar to ‘Means’).

In an industry case study, Eckartz found that Benefits Dependency Networks were considered “as being too complex, thereby creating a large burden for practitioners to actually use the method” [Ecka12, p.118]. However, the same practitioners later contradicted themselves by concluding that “the identification of dependencies [of resources and other outcomes for benefit] is essential” and...
that “it is further essential to make sure no conflicts occur” [Ecka12, p.119]. This indicates that constructing networks of cause and effect leading to a benefit (as in this subsection’s family of approaches) may appear to be a ‘bitter pill to swallow’, but retrospectively are considered worthwhile.

Finally, Sapountzis et al.’s recent comprehensive literature review on Benefits Realisation Management (BRM) [SaHK08] concluded that:

- Any BRM approach should consist of at least the four stages: Benefits Identification, Profiling, Planning, and Dependency Mapping [SaHK08, p.62];
- More research is required for the following aspects [SaHK08, p.63] (related to this thesis):
  - establishing traceability from benefit realisation plans (i.e., actions) to benefit outcomes;
  - making more robust and reliable predictions about the benefit to be realised;
  - understanding how to best utilise BRM to improve, and predict the effects of decisions;
  - clarifying the connections between BRM approaches and specific disciplines, such as IT.

Table 4.26: Feature Comparison Table (Key: ✔Explicitly, ✅Somewhat, ✘Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✔</td>
<td>✔</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of Approach #24: Thorpe’s Results Chain (Benefits Realisation Management)

**Summary:** Visualises the benefits of an initiative in terms of expected outcomes, and the outcomes expected from those outcomes. Unlike in goal graphs, outcomes are not restricted to those that the system’s agents are capable of causing.

**Pros & Cons:** A ‘bigger picture’ (wider reach than the ‘system’) is painted of an IT initiative’s benefits than with GORE. However, modelling is at the project (not requirement) level, contributions are qualitative, and uncertainty is ignored.

4.8 (Software) Systems Engineering Approaches

The final category of approaches to be reviewed are those that have some element of ‘means-end’ modelling for software or systems engineering, but do not fit in the previous categories.

4.8.1 UML Use Cases (& Cockburn’s Use Case Methodology)

The Unified Modelling Language is specified by the Object Management Group (the same owners of the aforementioned Business Motivation Model) as a standard object oriented language for software engineering modelling. The UML is a unification of the work of Rumbaugh, Booch, Jacobson, Coad, Shlaer, Mellor, and Wirf-Brock, among others [Fowl04]. Before the unification, each proposed their own approaches, leading to the infamous “methodology war” [BoEy97]. In other words, rather than “standing on the shoulders of giants”, the software engineering community was “stepping on the toes of giants” [Hamm80]. (E.g., the adoption of visual software engineering
languages was held up by different opinions on the shape of an object oriented ‘Class’: Booch used clouds, Yourdon used rounded rectangles, Objectory used circles, OMT used rectangles, and none were particularly motivated by heuristics [MoHM10]. Overall, the UML’s popularisation has meant that there is less room for misinterpretation and ambiguity in software engineering models [Rati97], although its usage in industry seems basic, and still subject to personal interpretations [Petr13]. Indeed, most usage of UML for requirements engineering focuses on the more basic diagrams [Petr13], primarily Use Case diagrams (which describe the actions that the system’s actors would like to perform using the system-to-be) and Class diagrams (which describe the objects in the system, their attributes, and their relationships with each other).

However, except for the functional requirements implied by Use Case diagrams, the UML has very little coverage of requirements, nor the ‘goal’ concept from GORE [Lams01] or generally the concept of value. On the other hand, UML’s extension for systems engineering (SysML), does have a ‘Requirements Diagram’ (RD) consisting of requirements (a specialisation of UML’s ‘Class Diagram’) and relationships between them (specialisations of UML’s $\langle<trace>>$ link stereotype) [SoVr08]. Nonetheless, this does not advance the state of the art much beyond Volere for the context of this thesis, since a SysML RD is essentially a ‘boxes and arrows’ visualisation of Volere requirement shells (where requirements are the boxes, and arrows are the traceability links). (SysML’s contribution is more significant when the context is bounded by the solution space, e.g., in requirements-design traceability.) Hence, numerous researchers have proposed integrations between UML, GORE and business modelling. For example Silingas et al. propose a mapping between the ‘Business goal’ and a ‘UML Use Case’ concepts [SiBu08], while Santander et al. propose the derivation of UML Use Cases from i* dependency models [SaCa02]. Finally, van Lamsweerde’s “From System Goals to UML Models to Software Specifications” [Lams09a] provides guidance on integrating KAOS and UML.

For this thesis’ problem, Use Case Diagrams are the most relevant of UML’s diagrams, and the methodology proposed by Cockburn for their construction is especially pertinent [Cock97]. UML use case diagrams consist of actors (stick men) and goals (ovals), and can be considered as a visual ‘index’ of the more comprehensive textual use cases advocated by Cockburn and guided by writing rules such as CREWS or CP (as evaluated by Cox et al. [CoPS01]). However, use cases are inherently solution oriented rather than problem oriented, given that they detail ways in which an actor would use the solution. Indeed, the use case goals are intended to be mapped directly to user-visible software functionalities, to later be operationalised by methods belonging to classes in object oriented programming languages. As such, ‘add item to shopping basket’ is a permissible use case goal, while ‘increase profit’ is not. (Indeed, Cockburn dissuades such ‘high level’ use cases with a use case ‘Sea Level’ metaphor [Cock01]).

Wasson recommends that use cases should be annotated with a numerical rating (1:low to 5:high) of ‘level of criticality’, along with ‘utility’, and ‘frequency of occurrence’ [Wass15, p.121]. The latter rating would go some way to making the stakeholder(s) think about the pertinence of the use case, and therefore might reduce the likelihood of implementing functionality that is never used. However, as has been previously noted, it is better to explain how the requirement would cause value, because, frequency of usage is not fully correlated with value (e.g., in the case of a ‘restore backup’ function).

In summary, use case diagrams provide hierarchical goal-refinement for user-visible functionality, whose refinements are described in terms of their cardinality, optionality, generalizability, and dependency. However, there is no relationship type in Use Case Diagrams to express that a use case is instrumentally valuable (i.e., contributes some extent of sufficiency) for realising a higher level goal or benefit for a stakeholder. SysML’s “satisfy” [SoVr08] link is the closest to such a link, but it
implies full sufficiency, is intended to have requirements rather than use cases as its destination, and
is not included in the significantly more widespread UML.

### Summary of Approach #25: UML Use Cases

**Summary:** Similar to Agile User Stories (Section 4.3.2) in that an end user’s desired uses of the software are described, but
UML Use Case diagrams visualise and describe constraints on hierarchical refinement of functional system goals.

**Pros & Cons:** UML Use Cases are effective at structuring software functionality goals from the user’s perspective, but they
are not treated as means to ends (to explain rationale), fostering ‘solution-eering’ rather than problem exploration.

### 4.8.2 Impact Estimation

Gilb proposes the ‘Impact Estimation’ approach to evaluate alternative designs against measurable
criteria [Gilb05a]. Essentially, a 2d matrix is constructed in a similar fashion to the aforementioned
House of Quality (Section 4.4.2), except for that ‘impact’ that an alternative is estimated to have
on a quality criteria is defined on an objective measurable scale.

For example, the criteria (i.e., goal) ‘Learning the system’ has a measurable fit criterion (a user
should be able to understand how to use the system within ‘10 minutes’, as opposed to the current
‘60 minutes’). Numerous alternative designs are then evaluated against the criteria, such as ‘On-
line Support’, whose impact on the ‘Learn the system’ criteria is estimated to be ‘5 minutes ±3
minutes’. As well as describing uncertainty through interval estimates, the Impact Estimation
approach describes the credibility (e.g., the rigour of the estimate and the reputation of the
estimator) in the estimate using a score within the [0,1] interval, (where 1 represents that there is
no doubt about the estimate’s validity, and 0 represents complete disbelief in its validity).

However, intervals are described without a confidence level, and so there may be high interpersonal
variability in reported intervals. Furthermore it is unclear which type of uncertainty is represented
(aleatoric or epistemic), and hence whether or not more information would reduce the breadth of
the estimate’s interval (epistemic uncertainty). The reasons for any variation in the estimate that
would not be reducible by further knowledge (aleatoric uncertainty) are also not described. Finally,
the effects of a non-functional requirement’s partial satisfaction are not explored, preventing analysis
of the effects of relaxing the requirement.

The qualities are defined using Gilb’s ‘Planguage’ requirements measurement framework [Gilb05a].
Practitioners often find difficulty in defining ways of measuring intangible software qualities that are
practical to use (i.e., ‘meters’ in Planguage [Gilb05a]), and at the same time, measure real qualities [Gilb07]. Hence, Gilb provides a collection of re-usable generic measurable software qualities for practitioners to use [Gilb05b]. However, to cite Basili et al., “‘improvement’ metrics may be anything from meaningless to dysfunctional if they aren’t related to the organization’s goals and to the questions about the organizations current state and evolving environment” [BoHZ05, p.376], and so “we cannot just use models and metrics from other environments as defined” [BoHZ05, p.98]. Indeed, the ‘Measurement Inversion Paradox’ [Hubb99] describes that most measured variables in IT projects have “zero information value”, while those with high information value are unknown and unmeasured.

Table 4.28: Feature Comparison Table (Key: {✔ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Means-End</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Benefit-Decomposition</td>
<td>☑</td>
<td>×</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>×</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement-Decomposition</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>×</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Benefits-As-Variable-for-Improvement</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-Functional Requirements</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Benefits-As-Magnitude-of-Improvement</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guide-Benefits-Elicitation</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Extent-Score</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Polarity</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Response</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Non-Linear</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Uncertainty-By-Unknowns</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Uncertainty-By-Variation</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contribution-Uncertainty-By-Propagating</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of Approach #26: Impact Estimation

Summary: Describes the contribution that alternative design options have on a set of quality criteria (as in the House of Quality), but in terms of measurable and verifiable metrics.

Pros & Cons: Verifiable estimates of a requirement’s benefit in terms of a measurable criteria’s scale are made possible (including uncertainty & credibility). However, the value of those criteria are themselves not explored as means to ends.

4.8.3 GQM+Strategies

GQM+Strategies is an extension of their Goal Question Metrics methodology, resulting from 30 years of its evolution and use [BHLM07]. Basili et al.’s aim in constructing the methodology was to enable traceability of software metrics at the project level (e.g., “Improve System Test Effectiveness”) to goals at the business level (e.g., “Improve Customer Satisfaction”), to avoid the aforementioned ‘Measurement Inversion Paradox’ [Hubb99].

Trendowicz et al. use GQM+Strategies to show the alignment of software project goals to business goals in the format of a 2d matrix [TrHS11]. The claimed benefits are that goals are defined quantitatively using a tried and tested metrics template, that metrics become ‘goal oriented’ (and hence less likely to provide practically useless information), and, that assumptions which underpin goal to goal contributions are made explicit, much like GRL’s belief element allows for. While the ‘Strategies’ element of the approach’s name introduces the notion of “a set of possible approaches for achieving business goals” [BHLM07], this, pragmatically speaking, adds nothing to GORE’s AND/OR goal graph refinement. Furthermore, the approach focuses on decisions at the project level [BHLM07, p.4], rather than the requirements level (answering the question of ‘which projects, rather than requirements, should be implemented?’), and so is not directly applicable to this thesis’
problem. Indeed, a large and variable number of goal abstractions (of different types, e.g., functional/non-functional) can be required to link a software requirement to a project goal.

In addition, contribution is not considered, and so when a link exists between two goals, the effects of the first goal’s satisfaction on the second goal are not explored. Thus, although each goal has a target satisfaction level (e.g., 5% profit increase), the impact that its child goals (e.g., software requirements) make toward it, (especially partial satisfaction), are not represented, along with estimate meta-data such as uncertainty, confidence, evidence, stakeholder agreement, etc. Finally, the approach lacks the rich concepts found in GORE which contextualise goals and support decision analysis, e.g., actors, obstacles, AND/OR refinement, etc.

Table 4.29: Feature Comparison Table (Key: {✔️ Explicitly, ✅ Somewhat, ✘ Not} Supported)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Means-End</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>✗</td>
<td>✘</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

**Summary of Approach #27: GQM+Strategies**

**Summary:** Disambiguates goals (making them verifiable) using a measurable goal template, and then links goals to other goals (sandwiched between ‘strategies’). Consequently, the pertinence of metrics is ensured.

**Pros & Cons:** Its goal prescription template enables unambiguous description of the benefits of a software requirement, but there is no notion of goal contribution, (and so full sufficiency with no uncertainty between goals is assumed).
### 4.9 Feature Analysis Overview & Summary

In order to summarise and conclude this section, Table 4.30 provides an overview of the capabilities (and the gaps) of the reviewed approaches.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Volere Requirements [RoRo12b]</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Agile Requirements [Cohn05]</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MODAF Traceability Matrix [Mini05]</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Software House of Quality [HeSP03]</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GRL [AGHM10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>GRL [AGHM10]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Automated Traceability [NRGB05]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Automated Traceability [NRGB05]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>NFR + GQM [DKPW07]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KAOS + Objective Gauge Variables [Lams09b]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KAOS + Probabilistic Layer [LeLa04]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KAOS + Utility Functions [FIKH11]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>KAOS + Confidence Factors [BoFH11]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------</td>
<td>---------------------------</td>
<td>------------------------</td>
<td>-----------------------------</td>
<td>------------------------------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------</td>
<td>--------------------------</td>
<td>----------------------</td>
<td>----------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td>---------------------------------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
<td>----------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRL + (Key Performance) Indicators [HBJY12]</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>GRL + Balanced Scorecard + SWOT [BJAM11]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>GRL + e3 Value Modelling [GoYR06]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>GRL + Consumer Values [ZdSG13]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>GRL + Stakeholder Agreement [HaAm15]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Goal Graphs + Business Value [CPYZ11]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Goal Graphs + Causal Dynamics [LoPr03]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>3G Alignment Framework [SiWo09]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>B-SCP (i* + OMG’s BMM) [BCVP06]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Balanced Scorecard + Regression [FuJe11]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>BRA Results Chain [Thor99]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>UML Use Cases &amp; SysML RD’s [Cock97]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Impact Estimation [Gilb05a]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>GQM + Strategies [BHLM07]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>This Thesis’ Proposed Approach (Chapter 5)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
This chapter has provided an extensive review of the approaches proposed in the literature relevant to the modelling and analysis of software requirements as instrumentally valuable to stakeholders. It is not surprising that the majority are goal-oriented either in name or in concepts, given the congruency between the concepts established in this thesis’ conceptual framework (Chapter 2), and the maturity of modelling techniques for representing goal graphs inherited from the AI field. This researcher’s impressions of the requirements engineering literature that was formed by the process of identifying and reviewing papers are consistent with those of other researchers in the value-based requirements engineering community, including:

- Aurum & Wohlin’s [AuWo07, p.114], Boehm’s [Boeh05], and Gordijn & Akkerman’s [AkGo03] view that the focus of existing RE methods tends to be value neutral, focused on cost and product quality instead of exploring the business case for technical decisions, despite that the purpose of RE is ultimately to add business value [Fava02];
- Bleistein et al.’s [BlCV06] and Singh & Woo’s view [SiWo09] that RE approaches concern operational goals (short-term transactional goals such as a user interaction goals) rather than strategic goals (longer-term goals that aim to create competitive advantage);
- Letier & van Lamsweerde’s view that subjective evaluation of a requirement’s contribution, as favoured by the majority of approaches, “violates a fundamental principle of requirements engineering calling for precise and measurable requirements and specifications” [LeLa04, p.3];
- Wieringa’s view that RE researchers tend to propose approaches or processes with toy problems, rarely evaluating their approach’s novelties for usefulness or usability [Wier05].

In the process of reviewing the literature for its suitability by the inclusion criteria, significantly more papers were read than were included. In general, it seems that very few approaches provide guidance to the requirements engineer on understanding stakeholder needs (i.e., ends that the software requirements are a means to) as instrumentally valuable for ‘something else’. Hence, most would generate requirements that would fail to be useful if stakeholder expectations about the ‘something else’ are wrong. Indeed, RE is intended to concern the derivation of solution requirements from problem requirements [Wier96], but those ‘problem requirements’ are merely ‘solution requirements’ to higher level problems owned by the stakeholders.

Numerous scholars stress the importance of requirements engineers being able to understand how proposed technology would impact business concerns [Bone11, SiWo09]. Yet, of the twenty seven approaches reviewed in this chapter that could model that a requirement is a means to an end, less than a quarter support refutable estimates (B2) of the degree of contribution, favouring often effortless but essentially meaningless subjective estimates (for the purpose of arguing the case for a requirement’s value, as opposed to say, documenting rationale). This is not unique to RE. For example, in the field of Decision Analysis, Howard writes: “Why, then, do inferior processes find favor with decision-makers? The answer is that they do not force you to think very hard or to think in new ways.” [Howa07, p.37].

Software tool support is also lacking in general, but this is not surprising given that researchers are motivated to publish ideas, rather than to produce operational tools. However, this means that a crucial aspect of the problem, i.e., finding, and re-using past models of a software capabilities’ value, (a problem highly suited to the advantages of computers over humans), is undeveloped. This researcher is reminded of a story about a train station that did not receive funding for refurbishment because it had very few users, but the users did not use the train station because of its need for refurbishment — a ‘chicken or the egg causality dilemma’.
When it comes to understanding the efficacy of the reviewed approaches on reducing the problems outlined in Chapter 1 (i.e., of redundant functionality and of poorly prioritised feature sets), no robust claims can be made for the majority. While the researchers sometimes provided anecdotal or qualitative evidence to argue for their approach’s usability and utility, this thesis’ author is not aware of any statistically robust investigations into their ability to reduce software engineering waste. The following quote from the ‘Future Work’ section of the INSTAL method’s paper (as mentioned in 4.6.2) is representative of the majority of ‘Future Work’ sections: “a more complete validation should be undertaken using empirical evaluations, and interviews of experts to explore the usability of the effectiveness of the INSTAL method.” [ThSa07]. This should not be surprising, since a longer timespan than is typically afforded by research projects is required in order to objectively evaluate the true effectiveness of value based software engineering methods. (The researcher must design and run a longitudinal study starting from a software project’s RE phase and ending after the resulting implemented software is used and its expected benefits are realised — which is in most cases years afterwards.) Indeed, only a handful of studies exist to support even the claim that requirements engineering in general is beneficial [BDFG05, DaCh06], (though many RE scholars claim that one reason why may be due to the ‘obvious’ need for RE [Lams01]). Nevertheless, the value of the discussed approaches has a strong logical argument: The essence of RE is understanding the problem and then specifying the optimal solution to it (such that risk of stakeholder dissatisfaction is lowered) [HuJD11], and most software engineering failures are due to the problem-understanding rather than the solution [Boch07, Jone94, Reif02].

Table 4.30 presented an aggregation of the feature analysis for each of the reviewed approaches, and in doing so, highlighted the gaps in literature in the form of capabilities unfulfilled by existing approaches. In summary, the approaches in the literature do not enable stakeholders to estimate and analyse the effect of a software requirement’s satisfaction on its related goals (and so creating benefit):

a) at varying levels of goal satisfaction extent (explaining the effects of partial/full requirement satisfaction on multiple levels of goal abstraction);
b) at varying levels of stakeholder confidence (explaining the extent to which a requirement’s satisfaction may not contribute to a goal as specified);
c) at varying levels of software usage (explaining the different profiles of soft-ware usage that could affect a requirement’s contribution to higher level goals);
d) at varying levels of stakeholder utility (explaining the non-linear relationships between the extent of a goal’s satisfaction and the stakeholder satisfaction to be gained);
e) at varying levels of stakeholder agreement (explaining the variance between the stakeholders’ estimates about the benefits that will be contributed);
f) with limited prior knowledge about the business goals (assisting the elicitation of the intrinsic values that the software features are intended to be instrumental for realising);
g) with tool support to describe, simulate, visualise, validate, communicate, re-use, recommend, and assist the modelling of a software requirement’s value creation mechanism(s).

The next chapter presents an approach that provides these capabilities, and hence by design satisfies each of the criteria that were defined in Table 4.1 and compared in Table 4.30.
5. Modelling the Instrumental Value of Software Requirements

“If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them.”

Henry D. Thoreau
The previous Chapter (4) concluded on the gap in the literature for an approach to modelling the instrumental value of software requirements that embodies the key concepts and principles (outlined in Chapter 2), and that supports the capabilities identified in partnership with this thesis’ industrial partners. This chapter presents a framework that aims to fully support those capabilities.
5.0 High Level Overview

An overview of the approach (originally defined in this author’s paper [ELDH12]) is that the benefits of software requirements are stated at various levels of goal means-ends abstraction modelled in GRL goal graphs, where a benefit is the advantage gained by treating a problem (and where goals can be inverted from problems). The degree to which benefit is contributed by each means is described in terms of contribution links, which are enriched with concepts from uncertainty and system dynamics modelling. Goal graphs are used as the underlying modelling framework because, as has been made clear by the literature review, they are well suited for describing and visualising: abstraction, refinement, dependencies, cause-effect, assumptions, as well as links to stakeholders and software components as agents. Hence, as well the approach being goal-oriented, it is agent-oriented as a consequence of the former, and somewhat object-oriented [Lam09a, p.278]. As will be shown throughout this section, these concepts closely map to those required for modelling the generation of instrumental value. Given the choice of goal graph languages, GRL [AGHM10] was adopted since:

1. GRL’s diagrammatic notation is amongst the most well known GORE metamodels within the RE community, primarily since it originates from the older i* language [AGHM10].
2. GRL was recently made an international standard through ITU specification Z.151 [Itu12].
3. Babar et al. concluded that i* is the most suitable GORE language for modelling Information System Strategic Alignment according to the ‘Strategy Map’ concept (as reviewed in Chapter 4) [BaWG11]. GRL was not included in this evaluation, but GRL inherits i*’s suitability.
4. GRL has an ontology describing its modelling concepts (where others are described informally) [Liu12].

5.0.1 High Level Process

There are numerous use cases for modelling the instrumental value of a software requirement (as discussed in Chapter 2.3, e.g., for requirements validation, trade-off analysis, etc.). Each use case includes the common process shown in Figure 5.1, where activity [7] is the main point of specialisation (i.e., the ‘Software Decision Question’ is the variable amongst the framework’s various use cases).

Figure 5.1: High level process (UML Activity Diagram). While the main directionality is modelled, it is not strict (and cannot be in reality), e.g., the process may loop (finitely) between [4] and [1].

The rationale for the activities and their order depicted in Figure 5.1 is as follows:

**Entry point** A software project is initiated, usually after rudimentary viability analysis.

1. Benefit is desired & generated for stakeholders, and so without first eliciting the stakeholders, no benefits can be elicited (but rather only guessed or inferred), and so [2] cannot occur.
2. Stakeholders will have expectations about the benefits (or dis-benefits) that the software
system as a whole will generate for them, usually before requirements are captured.

3. Some stakeholders will prescribe what the software system should do before thinking about any benefit that should be generated, but this should then trigger elicitation of benefits [2].

4. Benefits are traced to requirements in a goal graph, formulating the plan for their achievement, and to provide the structure for estimating requirement→benefit contribution.

5. The degree of goal-goal contribution needs to be estimated in order to understand the likelihood that the requirements will realise the benefits, and for impact (‘what if?’) analysis.

6. Ascertaining stakeholder agreement about the prescriptions (requirements and benefit) and the predictions (estimated degrees of requirement→benefit contribution) improves the likelihood for the model’s accuracy, as well as communicating the model to the stakeholders.

7. As a result of the predictions provided in the model, ‘what if?’ questions (e.g., if a requirement is not implemented?) can be answered, primarily feeding back to the adjustment of the requirements set, as well assisting the management of stakeholder expectations.

8. Accurate predictions about the benefits of software capabilities requires knowledge about the benefit previously realised by similar software capabilities in similar projects contexts.

(Exit point) The process is finished when the expected (or adjusted) benefits are realised.

There are no stipulations for the timing of design or programming work (and hence on adopting an Agile or Traditional development lifecycle). The goal graph constructed in activity [4] of Figure 5.1 may be scoped for the whole system, or one or many features (i.e., a release, backlog, epic [Cohn05], or other methodology-specific terminology). Modelling the complete system, or at least a complete release of the system, brings about advantages in verification of goal satisfaction (i.e., showing that the ‘root’ goal for system success is entailed by the set of requirements), interaction analysis, conflict management, resource allocation, and so on [Lams09a, p.35]. However, modelling smaller sets of functionality before implementation allows for requirements to be elicited from using the system rather than purely for the system [Fowl14], i.e., for requirements to be pulled as well as pushed. More often than not, this project management decision is constrained by culture, standards, or even legality [CIZ12, p.v]. Overall, the importance of value analysis (this framework) is increased where most of the system requirements are specified before development, due to the risk of their incorrectness.

5.0.2 Running Example Software Project: Digital Screen System

In order to illustrate the approach’s application, examples will be used in the context of a completed software development project familiar to the author. The software’s purpose is to create, manage and schedule both media files and live streams for digital signage (e.g., for advertising, live broadcasts, visual effects, etc.). The bespoke software was developed and implemented in a multiplex entertainments venue, whose system now consists of >60 screens, replacing printed posters.

This software project was chosen for the running examples for the following reasons:

1. The researcher played both ‘software developer’ and ‘requirements elicitor’ roles for it, and so had considerable knowledge of both the solution and problem domain.

2. The application domain is one in which most readers should be familiar with or aware of.

3. It should be possible to predict tangible benefits with reasonable confidence, since the software lies between the first two, and the most common of Zuboff’s {Automating, Informing, Transforming} IT classification (Chapter 2.1.3.5) from the operating venue’s perspective. (That it lies between the latter two from the advertising industry’s perspective is irrelevant.)

4. Numerous software and component-level decisions were made without any requirements analysis, which in some cases lead to significant re-work (and other costs, e.g. helpdesk tickets).
6. The examples are independent from this thesis’ industrial partners, allowing for this chapter to be free from Intellectual Property concerns, and hence for it to be viewable by all parties.

Each subsequent section will detail the core activities defined in Figure 5.1 as sub-processes (Section 5.1 corresponds to Activity [1] in Figure 5.1, and so on).

5.1 Eliciting Stakeholders

For a software system to be considered successful by its stakeholders, the benefits expected by them must be first elicited in order for those benefits to be realised, unless the software developers either happen to be unusually lucky [Jain07, p.81], or unless significant unexpected benefits occur. A stakeholder is defined as an “individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations” (including developers) [Ieee11b, p.6]. Figure 5.2 reduces this definition’s wide scope, which does not mean to say that the excluded stakeholders are no longer important in the software project, but rather that they are not considered essential in the exercise of modelling a software requirements’ instrumental value.

Omitting stakeholders (a classic mistake in software engineering [GaWe89]) leads to omitted expected benefits and requirements, leading to the development of software that would be unlikely to satisfy the stakeholders [Char05]. Interestingly, surveys on the influences on requirements selection in industry show that who proposed a requirement can be more important than what a requirement is [WoAu05], and so it is crucial that expected benefits are contextualised by who will benefit.

Numerous comprehensive but generic taxonomies of software project stakeholders are available, e.g., from the IEEE [Ieee11a, p.24], the SEI [Soft09], Volere [RoRo12b, p.10], and Alexander [Alex05]. However, to understand who is required for the activities depicted in Figure 5.1, a more specialised list of stakeholder roles is required. Such a list can be divided into the two primary groups: ‘Those who see benefit’, and ‘Those who see cost’. Of secondary but of no less important interest are ‘Those who inform the approach’. The roles within these three groups are enumerated in Figure 5.2.

<table>
<thead>
<tr>
<th>Stakeholder Roles of Interest to this Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S1.</strong> Those who personally mostly see benefit due to the investment in the software project:</td>
</tr>
<tr>
<td>a. <strong>Problem Owners</strong> will describe the problems or risks whose resolution (via the implementation, modification, or use of software) would generate direct benefit;</td>
</tr>
<tr>
<td>b. <strong>Problem Sufferers</strong> will describe the consequential problems or risks caused by those in S1a whose resolution would generate indirect benefit;</td>
</tr>
<tr>
<td><strong>S2.</strong> Those who personally mostly see cost due to the investment in the software project:</td>
</tr>
<tr>
<td>a. <strong>Enactors</strong> will be causally responsible for the realisation of benefit (e.g., end-users of the developed software, developers, process engineers, business analysts, etc.);</td>
</tr>
<tr>
<td>b. <strong>Approvers</strong> will be consequentially responsible for the realisation of benefit (e.g., project managers, sponsors, department heads, budget controllers, financiers, etc.);</td>
</tr>
<tr>
<td>c. <strong>Losers</strong> will need to be ‘bought into’ the shared vision or compensated so as not to counterattack or otherwise sabotage the software initiative — “you can’t make omelettes without breaking eggs”;</td>
</tr>
<tr>
<td><strong>S3.</strong> Those who mostly inform the approach (experts):</td>
</tr>
<tr>
<td>a. <strong>Target Setters</strong> will prescribe desirable and attainable targets for the software’s benefits;</td>
</tr>
<tr>
<td>b. <strong>Solution Designers</strong> will prescribe software capabilities to achieve the prescribed benefits;</td>
</tr>
<tr>
<td>c. <strong>Domain Experts</strong> will predict the degree to which software capabilities will realise benefit.</td>
</tr>
</tbody>
</table>

Figure 5.2: Stakeholder Roles of Interest (a stakeholder may fit into more than one category)

(Those who mostly see benefit (S1) refers to both benefit and cost as net, and so while they may be disadvantaged to some extent, the disadvantage is outweighed. Similarly those who see cost (S2)
would not likely cooperatively take part in the project if they only saw cost, but the majority of their compensatory benefits would not come from the software’s use, e.g., a software developer’s salary.) (The ‘Losers’ term used to represent those, who for example, may be re-skilled or made redundant by the software, may appear insensitive. However, it is standard terminology in VBSE [BoJa06, p.6].)

Membership to the roles is not mutually exclusive, since one stakeholder might play several of the roles in to varying degrees, especially in smaller organisations. (At the extreme, in a ‘one man department’ all roles might be played by one agent — though if there is no one else to communicate to, RE modelling approaches might not be as effective as hoped). However, this can be detrimental, especially where ‘Problem Owners’ prefer to play the role of ‘Solution Designers’. Hence, the ‘Expert’ stakeholder roles (S3) should be played by those most qualified for providing the required input, since politics and culture is often a detrimental influence in RE [MiMa12]. Adherence to the following criteria for selecting experts in elicitation activities is recommended:

1. Evidence or reputation of expertise;
2. Availability and willingness to participate;
3. Understanding of the problem area;
4. Impartiality (as far as possible — originally proposed for science, not engineering or business);
5. Lack of an economic or personal stake in the conclusions (as far as is possible);
6. Acceptability of costs and consequences of involvement;
7. Access to previous data and artefacts to be measured.

(Criteria 1-6 are from the field of Uncertainty Elicitation [OBDE06, p.29], while 7 was proposed along with others that overlapped with 1-6 by Basili et al. for selecting goal evaluators [BHLM07, p.9].)

Finally, to ensure completeness and even role distribution, coverage analysis comparing the elicited stakeholders to Figure 5.2’s roles would be useful. A robust assessment of the completeness of the elicited stakeholder set depends on the output of later activities (especially Section 5.2’s ‘Eliciting Expected Benefits’), since new output in those activities could render the stakeholder set incomplete. So at this stage, practical advice is to ask the already-elicited stakeholders ‘who else?’ questions (as is the essence of social network analysis approaches to elicitation [LiFi12]), and to make use of previous project experience (as is the essence of recommender systems [Lim10, p.116]).

5.1.1 The Boundary for Stakeholder Elicitation

Establishing the boundary for stakeholder elicitation in wide-reaching groups, such as those who would benefit as a consequence of the software’s use (S1b), is a trade-off between elicitation effort and coverage. Eliciting all of the stakeholders who would benefit would not be practically feasible, as is demonstrated by a ‘Reductio ad Absurdum’ (reduction to absurdity) argument:

i. Automation of manual tasks (such as removal of ‘expired’ advertisements from digital signage) would create more free time for the humans agents in the digital signage system;
ii. Some of those human agents spend their free time playing golf;
iii. Therefore, golf course owners should be elicited as stakeholders who would benefit as a consequence of the software’s implementation and use (stakeholder group S1b).

Just because golf course owners might actually benefit from the software, as in the ‘butterfly effect’ [Hilb04], does not make eliciting requirements and expected benefits from them worthwhile. The aim is not to produce ‘true’ models, but rather to produce useful models. Hence, whether a stakeholder is included in group S1 (those who see benefit) should depend on the criteria proposed in Figure 5.3.
Stakeholder Inclusion Criteria for ‘Those who see Benefit’ (S1)

C1. Are the stakeholders developing or funding the software (i.e., the ‘approvers’ and ‘enactors’ stakeholder group S2) likely to be positively or negatively affected if the stakeholder-to-be-elicted did or did not benefit?

C2. Would the expected value of the above effects (C1) not be insignificant, relative to the cost of eliciting and modelling them?

Inclusion criteria C1 is motivated by the fact that software projects are ultimately funded only because of their benefits for those making the investment, and so it is only worthwhile eliciting and modelling stakeholders whose involvement is contributory to the benefits desired by the investors. If these benefits are not realised, then the software project team might face negative effects such as discipline, damaged reputation, reduced business, etc., regardless of whether they were causally or consequentially responsible for realising them. (Indeed, causal responsibility [Somm07a] might not even be possible, since software often realises benefit from the improvement of processes owned by external stakeholders [Hamm01]. For example, an advertising brand manager might expect viewers of adverts played on digital signage to be interested in and purchase the advertised offers. The digital signage’s software developers could not realistically take responsibility for this according to the typical job role, but if that risk is not mitigated, e.g., by consumer analysts or graphic designers, it could negatively affect the venue’s perceived success of the digital signage system, and hence ultimately the software developers’ business.) Overall, eliciting stakeholders according to inclusion criteria C1 promotes the elicitation of traditionally ‘out of scope’ concerns, (given that in a typical RE process scope is limited to goals that the system’s agents are capable of satisfying without the cooperation of external actors [Lams09a, p.317]), and so helps to highlight vulnerabilities to the software’s perceived success.

Inclusion criteria C2 prevents the stakeholder groups from growing to unmanageable sizes, hence dealing with the previous ‘golf course owner’ ‘Reductio ad Absurdum’ argument. For example, according to inclusion criteria C1, one could argue that a particular company whose adverts would be displayed on the digital signage should be included as a stakeholder (e.g., since ‘CocaCola’ could be impressed with increased sales, and could recommend digital signage to other venues, who could choose to buy the developer’s software). However, the Expected Value (EV) (outcome multiplied by its probability) of that benefit would not likely satisfy inclusion criteria C2.

Finally, software intended to benefit very large stakeholder groups, such as government procured software intended to benefit a country, or for market driven [KDRN07] or Commercial Off The Shelf (COTS) software [NcMa99], are so-far unaddressed by the above guidelines due to the overwhelming size of the stakeholder set for those who would directly benefit (stakeholder group S1). Hence, the level of abstraction to which the stakeholders are elicited and modelled can be varied, as is discussed in the next section.

Verification against the Required Capabilities for the Approach (Chapter 4)

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

✔✔ ✔✔

[C1] Guide-Agent-Elicitation: “Provide guidance on eliciting the stakeholders who could gain benefits or costs from a requirement, or who could contribute to the requirement’s satisfaction.”

5.1.2 Modelling Elicited Stakeholders

The GRL’s actor subtypes [Liu12, Yu95, p.17] provide a framework for describing stakeholders at various levels of abstraction. For example, as Figure 5.4 shows, ‘James Smith’ can be described as:
[A-D] an actor (a generic entity to which dependencies can be ascribed), and/or;
[E] an agent (an actor with physical concrete manifestations), and/or;
[F-G] a role (an abstract characterization of an actor’s behaviour), and/or;
[H] a position (a set of roles usually played by one agent);

Describing stakeholders in this semantic richness enables a contextualised answer to ‘Who will benefit from the software?’, at various levels of ‘actor’ abstraction. This would be viable for localised software projects where the system is for one particular organisation (i.e., the ‘Venue Business’ [E]). However, in large projects, it would not be feasible to model all stakeholders as agents, unless modelling is restricted to a population sample [Alex05]. (It is unlikely to be able to even identify all ‘those who benefit’ (S1 in Figure 5.2) in software projects with “dozens of stakeholder groups and tens of thousands of users”, given that the majority of stakeholder elicitation methods overlook stakeholders [AlRo04]. Recently, stakeholder elicitation techniques such as intranets/wikis [Ubhi08], crowdsourcing, social network analysis, and machine learning have proven to be efficient [LiFi12].) Regardless of the project’s size, Yu argues that “modelling and analysis would be more intricate when these [actor] distinctions are introduced” [Yu95, p.23].

![Figure 5.4: Modelling Stakeholder Knowledge with GRL for the Screen System Running Example](image)

Project time is often scarce, and so the modeller should focus on modelling the most generic actor specialisation whose component actors each own the problem (instead of building a full social network model) for each problem to be solved by the software (i.e., each expected benefit). For example, if there is an inadequacy concerning the task of ‘Advert Dissemination’ [G] in the current system, it would not yet be efficient to model the agent ‘James Smith’ [E], since other agents playing the role would be equally affected. Conversely, it would be inaccurate to model the ‘Marketing Department’ [D] as the actor expecting benefit, since some of its composite positions such as secretaries would not be affected by the problem to a significant degree.

On the other hand, personal characteristics such as Holbrook’s consumer values [SMLB01] (e.g., a desire for excitement) might motivate new software capabilities. Such cases warrant the modelling of ‘John Smith’ [E] as an agent, since the consumer value would apply regardless of the roles the agent played within the system. However, care must be taken when modelling social stereotypes such as that ‘Graphic Designers’ [H] enjoy using the latest technology, since GRL’s link semantics permit the inference that all agents playing that position in the system desire that consumer value.

Two challenges become evident once the software project’s stakeholders playing the roles defined in Figure 5.2 have been elicited and modelled using GRL’s actor subtypes (as in Figure 5.4):
1. The example model in Figure 5.4 shows only one agent and an excerpt of their roles. In most real-world resource-constrained projects, such a model would grow to an unmanageable size. This problem can be divided into the two aspects of the model’s production and consumption:
   a. Problems in producing the model are primarily treated by the approach’s tool support (as is later discussed) for automated graph drawing. For very large diagrams, a matrix view of the model (similar to an $N^2$ stakeholder dependency matrix [Hitc03]) provides a spatially-compressed representation of actor relationships.
   b. Problems in consuming the model are primarily treated by the prioritisation of the modelled actors (and hence reduction of the model size via filtering) for the approach’s next activities. Populating a stakeholder ‘Power vs. Interest’ chart [Ieee11a, p.249] (otherwise known as an ‘Importance vs. Influence’ chart [FiPR06, p.153]) would be an effective technique for taking advantage of the ‘80-20’ Pareto rule [Boeh05, p.3] (i.e., that 80% of the important benefits are likely to be owned by 20% of the project’s stakeholders).

2. It might be difficult to communicate diagrams such as Figure 5.4 to stakeholders untrained in GRL’s visual notation, e.g., in ascertaining that shape [E] is an ‘agent’ (or even what an ‘agent’ is), given that “i*’s visual vocabulary is mostly semantically opaque” [MoHM10]. It is for similar reasons that UML (a considerably more ubiquitous software engineering modelling language than GRL) tends not to be used by the majority of industry practitioners [Petr13]. (However, note that the majority of practitioners are not trying to achieve high maturity software development — see CMMI [Sei12]). Despite this possible communication difficulty, it was considered better to adopt the existing international standard of GRL compared to creating a new notation (which would only create more need for training), or compared to using an alternative language such as KAOS (which does not visually distinguish between the various actor/agent types). Interestingly, an experiment comparing goal modelling languages by Matulevicius et al. found that i* (and hence GRL) was “easier to use” and indicated “a better overall quality for i* models” [MaHe07].

In order to make the GRL visual notation more accessible, the following tactics are suggested:
   a. Each element could be labelled with the natural language description, or, for large diagrams, a key could be added to the diagram (as is done in throughout this chapter);
   b. Icons could be replaced with a more semantically transparent set (e.g., a new icon for an agent: a stickman wearing sunglasses and a gun — agents of 007’s kind [MoHM10]),
   c. The diagrammatic notation could be compressed so that there are fewer notations to learn. E.g., elements not essential to the value model could be removed from the diagrams presented to non-technical stakeholders, e.g., all actor subtypes replaced by the actor type.
   d. All said, training is likely to be the most effective method for improving comprehension. Importantly, the reason to learn GRL must be motivated, e.g., by adopting GRL as an organisational standard. A reasonable estimate of the time taken to train a practitioner in understanding GRL models is no more than one hour.

---

**Verification against the Required Capabilities for the Approach (Chapter 4)**

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

- **C** Agents: “Represent and classify (e.g., ‘person X plays the role of Y’) agents’ [DalF93] (stakeholders, software components, etc.) related to a requirement or expected benefit.”

---

### 5.2 Eliciting Expected Benefits

This activity and the next (Section 5.3: Eliciting Software Capabilities) occur in an iterative (cyclical) and interleaving process, where the elicitation of an expected benefit may trigger the elicitation of a
software capability, and vice versa. In an ideal world, all software capabilities would be derived from an understanding of the problem to be solved. Indeed, the vast majority of ‘best practice’ processes for engineering [FoMo95], reengineering [Hamm01], problem solving [Wier09a], benefits management [PeWD07, p.9], and business improvement [Reif01] all follow the pattern of ‘Know current situation’ → ‘Determine where improvements are needed’ (i.e., benefits) → ‘Plan improvements’ (e.g., software), etc. (Nadler interestingly found a similar common structure in practice across diverse fields [Nadl67]).

However, in reality, numerous IT projects are oriented around interest in technology rather than problems (cf. Figure 5.5) [PeWD07], and so the previously mentioned sequence is reversed. In such cases, the difficulty of this benefits elicitation activity is increased, since problem exploration is now constrained to those solvable by the technology. In return, the usefulness of this thesis’ approach is increased. (Conversely, ‘Problem first’ projects do not guarantee that software requirements will not be wasteful, given that the understanding of the problem or stakeholder desires may still be faulty.)

As was defined in this thesis’ conceptual framework (Chapter 2.1.3), a benefit is “something that is advantageous or good” [Dict14] about “an outcome of change which is perceived as positive by a stakeholder” [Brad06] that could be realised “through the delivery of objectives” [Turn08]. Hence, the outcome of this activity is a set of objectives, that ultimately represent expected benefits only when they describe ‘something that is advantageous’ to a stakeholder about a software requirement or another objective. Nevertheless, in the initial iterations of this activity, ‘starting point’ goals must temporarily exist in isolation so that other goals can be abstracted and/or refined from them.

If a software capability has already been elicited, e.g., if ‘James Smith’ prematurely expressed a desire for the software capability: ‘the software should be able to automatically delete media files after a specified expiry date’, then this activity should start by eliciting the current or foreseen problem(s) affecting the various stakeholders that could be positively influenced by the operation of the proposed functionality. Otherwise, this activity should start by identifying problems that the stakeholders find ‘disadvantageous or bad’ (which will always exist according to the ‘Theory of Constraints’ [GoCo86]), and whose resolution would be within the scope of the software project. For example, ‘James Smith’ could be interviewed in the capacity of his ‘Advert Dissemination’ role about problems disadvantaging that role, such as that ‘removing expired adverts takes too much manual work’ (i.e., after advertised promotions are no longer on offer). Four obstacles are especially likely during this enquiry:

1. Stakeholders might claim not to be disadvantaged in any way by the current system due to not knowing better ways of working, and so:
   a. Stakeholders should be encouraged to think of problems in terms of outcomes rather than ways of working, e.g., knowing that ‘10 clicks to select a file to upload’ is fatiguing, instead of knowing that ‘drag and drop’ requires less clicks than a ‘browse dialog’.
   b. The requirements engineer could analyse current ways of working in order to suggest problems (which the stakeholder would then have to recognise, believe in, and then ‘expect’), e.g., via ethnographic observation [KoSo98], strategic document analysis.
[ThSa07], goal-oriented document analysis (identifying verbs such as, e.g., “achieve, avoid, maintain, improve, increase, reduce, make”) [AnPo98, p.164], value stream mapping [MaSB11], server activity log analysis, and other knowledge acquisition techniques [ByCZ92].

2. In greenfield software projects where a ‘system-as-is’ does not exist (as opposed to brownfield projects) [Lams09a, p.41], stakeholders may not yet be playing the roles that would disadvantage them (e.g., if the digital signage system was not replacing an existing way of showing media such as advertisements). In such cases, problems that would be solved by a software capability can be elicited by hypothesising scenarios of alternatives, (e.g., from the running example’s context: ‘the most likely substitute for ‘automatic media expiry’ is manual deletion, which is time consuming’).

3. Stakeholders passionate about having a software capability might claim to be disadvantaged simply by the lack of it in the current system, e.g., ‘automatic media expiry is not possible’. Instead, problematic outcomes associated with existing ‘ways of working’ must be elicited.

4. Not all stakeholders will find the same things problematic, e.g., some agents might find repetitive work (entailed by a role) a therapeutic respite from more mentally challenging work (entailed by a position). When agents provide conflicting views on the problems affecting a role, the importance of distinguishing between the ‘agent/position/role’ types is stressed.

Finally, if a stakeholder is not aware of any problems to be solved, then the requirements engineer could be assisted in eliciting an actor’s problems, by examining the various aggregated lists of values for organisations (1-2) and individuals (3-4) summarised in Table 5.1. Each list has been derived by their respective authors’ performing clustering exercises on other lists, and each claims to be all inclusive. (The main difference between them lies in the level of abstraction in which they are stated).

An organisation (represented by a GRL actor with ‘is part of’ links toward it) has non-personal problems that affect many positions, roles, and agents (e.g., ‘Finance’ in Table 5.1’s [1]). Examination of the organisation’s strategy [SiWo09] and its current situation (e.g., its ‘Strengths, Weaknesses, Opportunities, and Threats’ [Iiba09]) could also be useful for identifying ‘problematic states’. Roles and positions represent functional responsibilities within an organisation, and so tend to have operational problems (e.g., ‘Rework’ [2]) that affect agents playing them. Finally, human stakeholders (agents) tend to have desires of a personal or psychological nature (e.g., ‘Power (Status)’ [3] or ‘Security’ [4]). (Note that the values listed in Table 5.1 are intended to prompt the elicitation and further refinement of stakeholder problems, rather than to be exactly them).

Completely enumerating all of the problems a stakeholder would like to be solved by the software (i.e., building a ‘true’ model) is not the aim of this activity, since while omitted expected benefits might cause the lack of their operationalization and realisation by the software, so too might ‘analysis paralysis’ [Ambi12] (— where the requirements & design phases consume most of the project’s resources). Hence, in the initial application of the approach, problems should only be elicited on the condition that they would cause dissatisfaction with the software if they were not treated. In other words, only a portion of needs that could be ‘perceived’, should be ‘retained’ and ‘specified’ (and only some of those are likely to be ‘realized’ due to constraints) [PyOl13, p.222].
5.2.1 Modelling an Actor’s Expected Benefits

After a stakeholder’s problems-to-be-solved have been elicited, they are transformed to isolated expected benefits, defined explicitly, and then modelled in GRL, as shown by the three stages in Figure 5.6.

While each stage in Figure 5.6 is described as an explicit activity, an experienced modeller might choose to only define and model expected benefits at the third stage, i.e., to skip to Section 5.2.2. After these three stages, Section 5.2.2 ‘hardens’ the most important benefits to remove ambiguity.

---

26 The indicators within ‘{}’ are from a literature mapping exercise from each ‘Perspective’ to individual ‘IT benefits’ [Ecka12, p.123]. The majority of the example benefits are at the Operational & Tactical level, since a practical understanding of Strategic “vary widely across papers”, or are unidentified [EDWH09, p.5].
Stage 1/3: Defining a Problem as an Actor’s Belief

The first stage depicted in Figure 5.6 depicts the modelling of an actor’s belief about the problems that they wish to be solved. In doing so, the primary challenge is modelling the appropriate level of abstraction, in order to maximise the potential for eliciting more information about the benefit in the future stages of the process. For example, asking another stakeholder (perhaps ‘James Smith’s line manager) about the benefits of treating problem [1] in Figure 5.6, would be more likely to lead to better (i.e., mechanistic) descriptions of expected benefits, than if the problem was stated abstractly as ‘Too Much Manual Work’. Referring back to the analogy of a chain of instrumental value derivation from A→Z (Chapter 2.1.1), it is better for this ‘starting point’ problem to be closer to A than to Z, since otherwise the description of B-Y (and hence the description of how benefit is generated by reducing A) may be omitted. In other words, it is preferable to first elicit direct, low-level, and specific problems as beliefs.

One belief about a problem might be inherited by many stakeholders (i.e., agents) due to the one-many relationship between agents and roles or positions. For example, the belief about the problem ‘Removing Expired Adverts…’ (Figure 5.6) is explicitly owned by the ‘Advert Dissemination’ role, and so is inherently owned by the actors shown in the first row of Table 5.2.

Table 5.2: Actor Problem Ownership Matrix (✔ Explicit Ownership, ✅ Inherited Ownership)

<table>
<thead>
<tr>
<th>(Belief about) Problem</th>
<th>Role</th>
<th>Agent</th>
<th>Position</th>
<th>Actor (Aggregate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i. Removing Expired Adverts Takes Too Much Time</td>
<td>Advert Dissemination</td>
<td>James Smith</td>
<td>Graphic Designer</td>
<td>Marketing Department</td>
</tr>
<tr>
<td>ii. Revenues from Renting On-site Advertising Spaces are Low</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Contrary to as in traditional RE ‘problems vs problem owner’ matrices [Wier96, p.87], ideally, one belief about a problem should not be explicitly owned by more than one problem owner, due to the inheritance mechanisms afforded by the adoption of GRL’s actor meta-model. If this appears to be the case, then it suggests that:

Beliefs in i* & GRL are traditionally used to describe an actor’s rationale for design decisions, but it seems semantically coherent to use them for describing beliefs about ‘problematic states of the world’.

27 Beliefs in i* & GRL are traditionally used to describe an actor’s rationale for design decisions, but it seems semantically coherent to use them for describing beliefs about ‘problematic states of the world’.

5.2 Eliciting Expected Benefits
Stage 2/3: Inverting a Problem to Define an Isolated Expected Benefit (Qualitative)

The transformation from problems (modelled as GRL beliefs) to qualitatively defined isolated expected benefits (modelled as GRL softgoal), i.e., Stage 2 of Figure 5.6’s three stages, involves:

a) **Resource commitment**: in contrast to merely recognising a problem’s existence, goals imply that their owner will commit their own, or others’ resources in order to satisfy it (as was established in Chapter 2), and so the remaining process would not continue for some problems;

b) **Logical inversion** [Tooh18]: finding the positive version of the negative state of the world, e.g., negative state: ‘…takes too much work’ inverts to: ‘…does not take too much work’;

c) **Goal de-idealisation** [Bana10]: setting realistic expectations about treating problems, since they are rarely completely solvable (as is ideal), regardless of constraints & trade-offs.

The last step of goal de-idealisation is especially crucial, since it helps to avoid disappointing stakeholders who have unattainable expectations about the benefits of the software capability(ies). A fundamental aspect of goal de-idealisation is being specific about which aspect of a problem and how much of it will be solved, in order to allow solutions to the un-solved aspects (or extents) of it to be planned. For example, a stakeholder might prematurely propose the software requirement that ‘all user-uploaded data be encrypted using RSA with a 4096-bit encryption key’, with the belief that its benefit would be to ‘practically eliminate any chance of data theft’. However, as Figure 5.7 comically illustrates, encrypting data is not alone sufficient for avoiding ‘data theft’ 28, and so the expected

---

28 Ironically, more advanced computer security can ultimately disadvantage humans, e.g., the strength of biometric systems is a weaknesses against committed attackers due to its encouragement of body part theft.
benefit should be made more specific (e.g., ‘...minimise data theft via cryptanalytic attack’), which indicates that other (perhaps unaddressed) modes of ‘data theft’ exist that require mitigation.

Figure 5.7: The benefits of any software capability should be conceptually precise and not over-exaggerated, in order to be able to mitigate risks such as ‘$5 wrench’ attacks (comic © [Munr09]).

**Stage 3/3: Enrich an Expected Benefit’s Definition with a Variable to be Influenced**

The third and final stage depicted in Figure 5.6 transforms loosely-defined softgoals to become structured and variable-oriented, via conceptual disambiguation and definition by template.

**Conceptual Disambiguation** of an expected benefit minimises the scope for variation in stakeholders’ interpretations. Primarily, they should be defined in terms of a measurable variable\(^{29}\) to be either:

a) **Reduced** or **Increased** when the variable has a current value in the application domain, or;

b) **Minimised** or **Maximised** when the variable would only have a value after the system’s implementation. E.g., the variable ‘number of hours per week spent removing old adverts’ would not have a current value if there are no advertisements of any kind in the ‘as-is’ world).

The rationale for restricting a benefit’s influence (verb) to the above four, is that verbs such as Improve, Achieve, Keep, Ensure, Ease, Make, Maintain, etc. do not encourage measurable (and therefore specific) descriptions of expected benefits in terms of changes in some state of the world. They also do not infer an ‘acceptability direction’ [DoLo05], which dictates whether a higher or lower number on the benefit’s numerical scale is desired. In Anton & Potts’ seminal taxonomy of goals, expected benefits would primarily be classified as a type of Improvement goal (rather than as Achievement or Maintenance goals\(^{30}\)). Only in exceptional cases where an expected benefit absolutely cannot be described as a numerical change, should they be described as an Achievement or Maintenance goal; Typical non-exceptional uses would be to enrich the description of an earlier described numerical change, e.g., to use a stakeholder’s own terminology. For example, perhaps a Reduction in ‘Manufacturing Lead Time’ might Achieve alignment with ‘Planned Product Release Schedules’, or an Increase in ‘Customer Base’ might Maintain the ‘#1 Widget Provider in the UK Position’. (Such modelling of contribution between benefits is guided later in Section 5.4).

**Explicit definition** of each expected benefit is required so that it can be recorded, completeness-checked, communicated, and ultimately re-used. While a myriad of semi-structured natural language goal description templates exist in the literature, e.g., [AnPo98, ClBa10, CoPS01, DaLF93,  

---

\(^{29}\) A reader sceptical about the ability to measure intangibles is referred to the arguments and guidance within [Gilb05b, Hubb10, Maha10, Mclu03, p.3]. The rationale for measurability was justified in Chapters 2 & 4.

\(^{30}\) GB_RAM’s other categories: Maintenance (‘sustainment of some existing condition [always]’) & Achievement (‘making certain that a particular state is achieved [eventually]’) [AnPo98, p.164]. KAOS shares this taxonomy [Lams09a, p.268], and GQM+Strategies’ has similar (Maintenance/Success/Growth) [BHLM07].
RoSA98], fewer exist for goals involving changes in quantities, e.g., [BHLM07, ClBa10, p.75, Gilb05a]. Of those, Basili et al.’s GQM+Strategies goal formalisation template [BHLM07, p.11] is the most comprehensive and modular, and so it is adopted and customised here\textsuperscript{31} for defining individual expected benefits. While Basili et al. intended for all eight of the template’s fields to be populated, this framework proposes a two-stage process, whereby initially, only three of its fields are populated, as shown in Figure 5.8. (The later-populated remaining fields detail the quantitative prescription, which, as established in Chapter 4, may not be necessary for every expected benefit, but are crucial for disambiguating those which would significantly affect the perceived success of the software, e.g., those owned by the most powerful or influential stakeholders).

![GRL Model of Expected Benefit](image)

**Figure 5.8: Defining an Expected Benefit in its initial ‘soft’ stage (so far, only three of many more fields are elicited, such that it constitutes what GRL would consider a softgoal).**

The ‘Owner’ field is populated by a member of the ‘Problem Owner’ stakeholder group (S1.a), and its value is inferred from the GRL actor whose actor-boundary encapsulates the expected benefit. (This fundamental field is not in the GQM+Strategies template, but is in PALM’s [ClBa10].)

The ‘Activity’ field’s value is restricted to one of \(\{\text{Increase, Reduce, Maximise, Minimise}\}\) (in non-exceptional circumstances, as previously described), and provides the Verb for the ‘Verb[NounPhrase]’ label in the GRL diagram.

The ‘Focus’ field defines the property, aspect, or attribute of the ‘Object’ to be affected by the activity. This avoids ambiguously prescribing a desired increase in ‘profits’, for example, without specifying ‘of what’ (‘product X’, ‘company X’, etc).

### Verification against the Required Capabilities for the Approach (Chapter 4)

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

- ✔️ [C3] Agent-Desire: “Be able to represent that agents desire the achievement of an end (i.e., to answer ‘who benefits from the requirement’s implementation?’).”

This approach proposes an object-oriented view of ‘Expected Benefits’ in order to ensure consistency (and hence machine-readability and re-usability). As such, values of the ‘Object’ field are relatable to either a UML Class or Object, depending on whether the benefit concerns a type of thing (Class), or one or more instances of an entity type (Object). The differences are as follows:

\textsuperscript{31} Important fields from the other templates have been added to the GQM+Strategies template (signified by the * symbol), some have been adapted, and some have been removed. Two fields from the GQM+Strategies template are omitted: ‘Constraints’ and ‘Relations’, since these can be defined in the goal model (in GRL) using obstacles and links (e.g., dependencies), respectively.
An ‘Object’ field that is related to a UML Class:

**Concerns:** All possible (& future) instances (bounded by later-specified ‘Scope’ & ‘Timeframe’ fields);

**Form:** The sentence “An instance of this Class is [a/an] ‘Expired Advert’” should parse [AgGO13];

An ‘Object’ field that is related to a UML Object:

**Concerns:** Instance(s) of known thing(s), e.g., an existing ‘Media Library’ in an application domain;

**Form:** The sentence “[This|These] <‘Media Library’> [is an|are] instance[s] of a Class” should parse;

(The ‘Object’ field in the ‘Expected Benefit’ template is not renamed even if it refers to a Class, since ultimately, any benefit will be realised by (and checked by) changes to instantiated Objects.)

It is tempting to infer too much from the plurality of the ‘Object’ field, e.g., by assuming that ‘Expired Advert’ refers to a UML Object, and ‘Expired Adverts’ refers to a UML Class. However, a particular instance is neither indicated in ‘the removal of an expired advert’ nor in ‘the removal of expired adverts’. Rather, the most significant possible inference is that the latter’s (later defined) measurement scale might be expected to refer to time spent in total (e.g., *per month*) removing adverts, whereas the former’s scale might refer to the average time spent removing an advert (e.g., *per removal*).

When defining the ‘Object’ and the ‘Field’ attributes, adherence to Object Oriented naming conventions [AgGO13], as advocated by industry guidelines [Micr10], is recommended. Primarily, both fields should be either nouns or noun phrases, enabling their composition to form a ‘possessive noun-noun compound’ [HeMP11, p.1122], where the focus ‘belongs’ to the object, e.g., ‘<Focus=Time Taken to Remove><Object=Expired Adverts>’ → ‘Time Taken to Remove Expired Adverts’). Better grammatical composition might be achieved by swapping the order of the ‘Object’ and the ‘Focus’ fields in the parsed sentence, permitting the more succinct ‘<Focus=Removal Time>’.

The GRL element label should then follow the pattern of ‘Activity[Object Focus]’ or ‘Activity[Focus Object]’.

Where benefits concern subjects with complex names, e.g., in a reduction of ‘Car Part X Manufacturing Lead Time’, then understanding which part of the name is the Object, and which is the Focus field can be confusing, since the Object could be the *Car*, the *Car Part X*, or the *Cart Part X Manufacturing (Process)*. The GQM+Strategies literature (which first proposed the ‘Object’ and ‘Focus’ fields) offers sparse guidance on this, merely describing them as “the object of measurement”, and “the main focus of the goal” [BHLM07]. Hence, this framework refers to the Object Oriented naming convention that the sentence: “a <Class> has [a/an] <Attribute>.” [AgGO13, p.3] should be coherent (applied to the ‘Object’ and the ‘Focus’ fields, respectively), as well as to the ‘God Class’ Object Oriented Anti-pattern [LiPo09]. As an example, “a <Car Part X> has a <Manufacturing Lead Time>,” is more logical than “a <Car> has a <Part X Manufacturing Lead Time>,” since the latter implies that ‘Part X’ is a property of a car, instead of being part of the car’s composition. Further, due to a Car’s high number of composite classes (other parts), the latter sentence would cause the <Car> class to be considered a ‘God Class’, which is detrimental for ‘Coupling & Cohesion’ [Wier96, p.12], and hence for later information re-use. (A ‘God Class’ is one that does or knows too much, and so it should ideally be split into other classes [SmWi00].)

Finally, guidance on eliciting the ‘Object’ and ‘Focus’ fields for the ‘Expected Benefit’ template can be found by exploring the problem domain’s UML models. For example, Figure 5.9 shows excerpts of UML models pertinent to the Expected Benefit modelled in Figure 5.8 (pertinence is denoted by: *).
5.2 Eliciting Expected Benefits

Expected Benefits can be elicited from UML models (especially in ‘Technology First’ projects) by asking how advantage would be created by:

- Initiating one of the use cases (which implicitly examines the value of a Class’ method(s));
- Transitioning an object’s state, e.g., changing an ‘Advertisement’ from ‘Expired’ to ‘Removed’.

It is not unreasonable to imagine that UML models of the problem domain (e.g., at least of its classes, relationships, and perhaps states) will already exist in high maturity organisations, who often model their project’s domain knowledge in ontologies (often represented in UML) [SSLM11]. Otherwise, comprehensive guides for modelling the system-to-be with UML for RE exist [Fowl04], and doing so is considered good practice [Petr13] regardless of this approach’s possible use of them.

5.2.2 Harden the Definitions of Expected Benefits (Softgoal → Hardgoal)

So far, an expected benefit (as in Figure 5.8) is ‘soft’, in that it is not unambiguously verifiable or falsifiable. Consequently, understanding whether the ‘Time Taken to Remove Expired Adverts’ has been ‘Reduced’ faces two obstacles. Firstly, it is not clear which ‘Expired Adverts’ should be evaluated, nor over which time period the ‘Time Taken to Remove’ them refers to. Secondly, any (e.g., a 0.01%) ‘Reduction’ is a valid reduction and hence implies that the benefit is realised (i.e., fully satisfied). This severely limits the ability to examine the consequential benefits of realising a benefit, as well as to examine the extent of contribution that a software capability makes toward it, since ‘extents of realisation’ does not make sense without first determining what ‘full realisation’ is.
Hence, the final stage in this activity (5.2) is the population of extra fields in the ‘Expected Benefit’ definition template to specify the benefit’s clear-cut satisfaction criteria, such that it can eventually be considered as a hardgoal in GRL (represented by a rounded rectangle in GRL’s visual notation). These extra fields consist of those remaining from the ‘GQM+Strategies’ template, as well as several fields considered important from other templates (denoted by the * symbols in Figure 5.10).

<table>
<thead>
<tr>
<th>GRL Model of Expected Benefit</th>
<th>Expected Benefit Definition Template</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Owner</strong>&lt;br&gt;Advert Dissemination (Role)&lt;br&gt;Reduced Time Taken to Remove&lt;br&gt;Expired Advert&lt;br&gt;Target: 6h 50m [reduction]&lt;br&gt;Threshold: 6h [reduction]&lt;br&gt;As-Is&lt;sup&gt;34&lt;/sup&gt;: 7h 10m&lt;br&gt;Scale: Hours elapsed between the expiration of an advert, and its removal, on average per month&lt;br&gt;Timeframe: 3 months after system deployment&lt;br&gt;Scope: All adverts, no matter how out of date, displayed on any signage site belonging to the venue&lt;br&gt;Author: John Doe (Requirements Engineer)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.10: Hardened ‘Expected Benefit’ template, such that its satisfaction is ‘clear cut’ (through quantified prescription), and so it can be considered as a ‘hardgoal’ element in GRL.

The four additional fields added between Figure 5.8 and Figure 5.10 are the Scale, Magnitude, Timeframe, and Scope. (The scale is not listed first in Figure 5.10 to promote the template’s readability, as in: “<Owner> expects the <[Focus/Object]> <[Object/Focus]> to be <Activity> by <Magnitude.Target>, as measured on the scale of <Scale>, to be achieved by <Timeframe>, within the context of <Scope>”. The GRL diagram label for hardened Expected Benefits should follow the syntax ‘Activity/Object Focus/(Magnitude)’<sup>35</sup>.)

The subsequent sections will discuss the population of the four aforementioned additional fields.

**Stage 1/3: Describing the ‘Scale’ of Measurement**

The first field to be specified is the ‘Scale’, which provides a measurable specification of the variable implied by the aforementioned ‘Focus’ field, and is later to be used by future estimates and evaluations of a benefit’s realisation. (The ‘Scale’ field is adopted from the Planguage quality requirements template [Gilb05b] — its “most important concept” according to Maiden [Maid06]. Such an explicitly numerical definition of the ‘Focus’ is not provided by the GQM+Strategies template.) The ‘Scale’ should be defined as one concise sentence, detailing the numerical variable’s unit of measure (e.g., hours, Likert scale points, £, etc.), which should not be dimensionless (e.g., %) because:

i. ‘concrete numbers’ encourage an understanding of the as-is state. For example, a ‘10%’ reduction in something can be too-effortlessly prescribed without thinking about its consequences or achievability, as compared to a ‘1 minute’ reduction in a ‘10 minute’ process.

<sup>34</sup>‘As-Is’ is swapped with ‘As would otherwise be [without the new software initiative]’ when the ‘Scale’ variable does not yet have a value in the application domain. For example, suppose that this digital signage system is not an upgrade over an existing system, i.e., there are no advertisements to be in need of removal.

<sup>35</sup>The Magnitude could optionally be the calculated percentage change from the As-Is value to the Target (as it is in Figure 5.10), and the Object and the Focus could be swapped, as was discussed in Section 5.2.1.
ii. human comprehension of ‘natural frequencies’ (e.g., “1 in 8 people”) is superior to dimensionless ‘abstract numbers’ such as traditional probabilities (e.g., ‘12.5%’ or ‘0.125’) [GiHo95].

A stakeholder can benefit from a change in a ‘fundamental unit’ (otherwise known as a ‘base unit’) comprised of one dimension (e.g., ‘time taken to remove [a particular instance of] an expired advert’), or by a change in a ‘compound unit’ comprised of more than one dimension (e.g., ‘expired advert removals per hour’). (The distinction between ‘compound units’ and ‘fundamental units’ is clarified in [Eshb09, p.57].) In systems dynamics terminology, the former ‘fundamental units’ represent stocks, while ‘compound units’ comprised of a time dimension represent flows [Ster00, p.141]; Stocks have a value at each moment in time (e.g., someone’s bank account balance on 1/1/2001 at 1pm), whereas flows change the value of a stock, and are measured over an interval of time (e.g., £’s per month deposited into a bank account).

The main complexity in defining a Scale for an Expected Benefit (unaddressed by Planguage due to its different intended context), is deciding whether the stakeholder’s problem is considered solved (i.e., benefit is realised), when either Stocks, Flows, or a Summary Statistics are influenced.

For example in the context of Figure 5.10, the following ‘Scale’ field values may seem viable:

a. **Fundamental unit (Stock):** e.g., the total number of [seconds | minutes | hours | days] elapsed removing all expired adverts (measured on a certain date implied by the later-defined ‘Timeframe’ field);

b. **Compound unit (Flow):** e.g., the number of expired adverts removable per [second | minute | hour];

c. **Compound unit (Summary Statistic):** e.g., the [seconds | minutes | hours] elapsed removing each expired advert, on average per [day | month | year];

d. **Compound unit (Aggregate Summary Statistic):** e.g., the total number of [seconds | minutes | hours] elapsed removing expired adverts, on average per [day | week | month | year];

The trade-off made when selecting one of the above measurement unit representations is primarily between the richness of the problem description, and the difficulty of estimating and evaluating effects on it. E.g., estimating values for (c) will be cognitively easier than for (d), since (d) is the result of a calculation requiring another estimate on the number of expired advert removals. Overall, (c) and (d) represent the most likely ‘ideal’ units of measurement for the majority of expected benefits, since:

(a) is not ideal since it depends on the estimator knowing how many expired adverts are (or will) be removed, and hence is highly specific to a particular context;

(b) describes a possible rate of expired advert removal (just as in ‘miles per hour’), but just because 60 adverts could be removed every minute, does not mean to say that there will be 60 adverts to remove, nor that every removal will take the same amount of time (i.e., follows a uniform distribution);

(c) may be preferable to (b) since it acknowledges variability and is less ambiguous in the measurement timeframe;

(d) may be preferable to (c) since it enables a description of there being more benefit to automating frequently performed activities than rarely performed activities.

This conclusion is supported by extensive research on expert elicitation of parameters for regression models (effects of X on Y, where Y is the Expected Benefit in this case). Garthwaite and Dickey advise that experts should be asked about the median value of Y, if a large number of observations were taken at the single value of X, (Ŷ is also the mean under the often held assumption of normal
distribution) [GaDi88]. More recently, O’Hagan et al. summarise their support for estimating averages, since they are “quantities to which people can relate” and “an expert can give assessments about [the average of] Y, without the need to consider random error” [OBDE06, p.142]. (Given that it is easier to estimate (c), but that (d) is more descriptive, it would be good practice to eventually describe both as individual expected benefits, where (c) is initially described and then linked to (d), following the later guidance on creating a contribution link between expected benefits in Section 5.4.1.)

The modeller should choose an appropriate duration of time for Scales — e.g., choosing ‘on average per month’ instead of ‘on average per day’, where ‘appropriateness’ is that:

- The duration of measurement should be wide enough to capture variation and trends, e.g., if a problem tends to occur every other Friday, then ‘per week’ is not appropriate.
- The measured quantity should be countable as opposed to calculable, e.g., ‘per hour’ is not appropriate if its value could only be decided by dividing from, e.g., ‘2 per day’. I.e., if a non-integer estimate is provided on an integer scale, then the scale’s duration of time should be modified to be able to naturally accommodate that estimate.

Finally, numerous statistical summaries (e.g., mean, mode, median, standard deviation, confidence/credible intervals, etc.) [OBDE06, p.5] can be used for describing an uncertain quantity, i.e., in (c) and (d). The importance of this is illustrated by the simple example that the set of response times [2,2,58,58,58]ms has the same mean response time as the set [30,30,30,30,30,30]ms, yet the latter set would be preferable in most software systems (due to consistent performance).

**Stage 2/3: Describing the ‘Target’ on the Scale**

Secondly, the ‘Magnitude’ field from the GQM+Strategies template is modified to expect at least three values as opposed to one (i.e., just the ‘Target’); Essentially, the ‘Magnitude’ field provides ‘points’ on the previously defined ‘Scale’, that define:

1. the current (As-Is) extent of the benefit’s satisfaction (can be null in Minimise/Maximise);
2. the lowest-acceptable (Threshold) extent (lower values cause complete dissatisfaction);
3. the expected (Target) extent (equal or higher values cause complete benefit realisation);

Without knowing the as-is value of the existing ‘way of doing things’, it is not possible to know if any change has been made, and hence whether any benefit has been realised. Therefore the ‘As-Is’ value should only be left undefined if the ‘Object’ and/or ‘Focus’ does not currently exist, i.e., where the expected benefits underlying problem would only exist after the system has been deployed. This would be common in green field system development where the implementing-organisation does not currently have an ‘as is’ system, (in the context of the running example: if the organisation did not currently have any form of advertisements to be removed). In such cases, the ‘as is’ value can be substituted for an ‘as would be [after the system deployment]’ value in order to represent the size of the expected problem (that, hopefully, a software capability would solve). Such ‘baseline’ values are important, since they act as a ‘reference condition’ for the causal proposition (see Section 2.1.2.1), and provide a number to be increased or reduced by software capabilities. Finally, if there is uncertainty about the ‘As-Is’ value, it may be specified as a confidence interval, e.g., ‘80% confident the as-is value lies between 5 and 8 hours’.

---

36 As influenced by Planguage’s {Past, Worst, Planned} fields [Gilb05b], respectively. Gilb also proposes ‘Best practically possible’ and ‘Competitor’ points, but only the first and the last are considered crucial.
Setting well-balanced ‘Target’s (ii) is the most challenging of the three, since they determine the minimum benefit required from proposed software capabilities, and can be responsible for the removal or addition of features and qualities. The ‘Target’ should be achievable within the limits of current technology and project constraints, otherwise the stakeholders would never be satisfied by the software. However, it is better to start with perhaps unachievable ambitions and gradually reduce (‘de-idealise’) them in light of new information gained as the project progresses. (Hence early-defined ‘Targets’ must be mutable given that the Cone of Uncertainty [Boeh08, fig.2] is unavoidable. For this reason, it is unfortunate that Targets also imply a commitment, given that they will likely be used for early cost-benefit analysis for the project’s funding.)

**Stage 3/3: Describing the ‘Timeframe’, ‘Scope’, and ‘Author’**

Thirdly, the ‘Timeframe’ field specifies when the ‘Target’ point on the ‘Scale’ needs to be achieved, since benefits of software are rarely realised instantly after the software’s implementation, e.g., due to users learning how to use the software (often taking months to reach its full potential [Thor99, p.89]). Then, the ‘Scope’ field provides the boundary for the ‘Object’ and ‘Focus’ fields, in order to disambiguate which ‘Objects’ will be affected by the ‘Activity’ (it cannot refer to every possible ‘Object’ in the universe). Finally, the ‘Author’ field records who proposed the expected benefit, which is important for distinguishing where, for example, a software engineer has proposed a benefit for an ‘Owner’ based only on a software requirement and their (perhaps incorrect) understanding.

**Verification against the Required Capabilities for the Approach (Chapter 4)**

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

✔ [A7] Benefits-As-Magnitude-of-Improvement: “Provide guidance on representing benefits in terms of unambiguous, measurable, time-specific, benchmarked, and stakeholder-owned objectives (as improvement to its variable in the application domain).”

### 5.3 Eliciting Software Capabilities

The output from this activity should be a set of functional capabilities and non-functional qualities that comprise the requirements on the software system. Later instances of this activity (instantiated via a feedback loop in Figure 5.1’s process) entail modifications to the composition of this requirements set (i.e., the selection, order, and structure of requirements) and to its members (i.e., changes to individual requirements).

A wealth of established software requirements elicitation techniques have proven to be useful\[37\]. Hence this section will instead focus on guiding the representation and organisation of software system requirements, such that they are optimally defined and structured as input for the approach. If the approach is used after software requirements have already been elicited\[38\] (and so where the requirements set already exists in a similar fashion to that specified in this section), then the modeller may skip to the modelling stages (Stage 3 of 5.3.1 and Stage 2 of Section 5.3.2).

---

37 Comprehensive taxonomies of elicitation and specification techniques are provided in [HiDa03, YvWi05], as well as numerous comprehensive textbooks [Alex02b, Kovi98, Lams09a, RoRo12a, SoSa97b].

38 E.g., where the approach is used for software requirements validation, or informing trade-off analysis.
5.3.1 Modelling Software/System Features

Initially, the requirements set of interest is comprised of high-level software functionalities, without which the software would fail to ‘be’ the software that the stakeholders initiated the project for. Mavin refers to such a set of high level requirements, as a ‘does the boat float?’ requirements set (i.e., if the ‘must float’ requirement fails, then it is not a boat) [Mavi14], while Lausesen simply refers to a ‘top-ten requirements list’ [Laue02]. More formally, at this stage we are interested in requirements at the ‘Feature’ level: a “distinctively identifiable functional abstraction” or the “end-user’s perspective of the capabilities” [KCHN90, p.28] of a software system that must be “implemented, tested, delivered, and maintained” [CIHS08]. More simply, “a feature is an abstract description of a requirement, and it is detailed by one or more functional requirements” [DeNP12].

The rationale for initially focusing on features instead of lower-level functional requirements (e.g., event-oriented requirements [MWHN09]), is that efforts spent analysing low-level intricacies or mechanics of a feature would be wasted if the assumed value of the feature as a whole was misperceived. Notable exceptions (and so ideal requirements for the approach), are where:

- A feature is expected to be particularly costly, or would lead the development through a narrower and practically irreversible path, e.g., if it would affect the software architecture;
- Decisions about alternative feature need to be made, and/or documented (rationale);
- There is uncertainty (acknowledged or not) about the need for, or the meaning of a feature.

Stage 1/3: Structuring Software Features (Functionality)

The first of this activity’s three stages involves creating a structural breakdown of the proposed software features, as represented by the Feature Model in Figure 5.11. (A comprehensive guide for Feature Modelling can be found in [PoBL05].) Ideally, these features should be derived from the expected benefits defined in Section 5.2 (and/or their various levels of abstraction, as are later defined in 5.4). However, a Feature Model might already exist due to premature solution design, or where a pre-developed software package is to be customised rather than developed (e.g., for COTS).

The hierarchical nature of a Feature Model aids the selection of the suitable level of a software capability’s abstraction for analysing its benefit. Interestingly, since the ‘root’ level feature (i.e., the whole software) is composed by its lower level features (along with decisions made on OR/XOR/Optional refinements), the expected benefits of the root could be stated equally to those of a leaf. For example, [LEAF2] (Figure 5.11) might be expected to ‘Reduce Time Taken to Remove Expired Adverts’, but then so would [ROOT1]. However, the following differences apply:

- The contribution from [ROOT1] to the expected benefit would be from all of the proposed features, rather than just from [LEAF2]. Hence, the extent of the contribution to the benefit might be different, because many features might additively contribute to one benefit.
- The requirements engineer might not have claimed the expected benefit of [ROOT1], had [LEAF2] not been presented in the feature model, since a person’s mental model of the features is not likely to be as complete as an explicit model.
- The usefulness of a model showing a contribution from [ROOT1] to the expected benefit would be significantly lower than if it were modelled from [LEAF2]. For example, the requirements engineer would not be able to use the resulting model to answer ‘what advantage would not be provided to the stakeholders if this feature were not implemented?’. In other words, the mechanics of expected benefit realisation are lost by modelling at high feature abstraction.
Stage 2/3: Defining Software Features

The second stage is the production of a textual definition of the features (e.g. shown in Figure 5.11). Representations of required features can range in rigour from mathematical formalisms (e.g., in ‘Z’) to unstructured natural language (e.g., ‘user stories’), depending on the risks of incorrect specification. From this approach’s point of view, a software feature is adequately defined when an unambiguous specification exists for verifying whether it has been adequately implemented or included in the system. Without this, reasoning about the benefit that would be caused by it could be useless due to the high possible number of interpretations of what it means for [LEAF2] to be implemented.

After reviewing the numerous requirement metadata schemas proposed in the literature, both Firesmith [Fire05] and the INCOSE [HuJD11, p.81] propose the ‘Verification Method’ field as mandatory, whose purpose is to provide this unambiguous specification. Similarly, the Volere requirements shell proposes a ‘Fit Criterion’ field [RoRo12b], stressing the need for its measurability (in the worst case on a Boolean scale). For example, feature [LEAF2] (Figure 5.11) might have its Fit Criterion specified as ‘Any media item within the media library whose date/time for expiry has been set by a user, should not exist in the media library after 60 secs after the passing of that date/time’.

Stage 3/3: Model Software Features as Software Agent Capabilities

The final stage is to model the proposed software features as capabilities of the software-to-be agent. A software requirement could be modelled from other actors’ perspectives (e.g., as something a software developer will implement, something an end-user will use, etc. — as discussed in Chapter 2.1.6.4). However, in most use cases for this approach, we are interested in the effects of the software-to-be in use. As such, examining the resource dependencies of a software requirement as a development task would likely result in less-relevant models, e.g., concerning the developer’s environment.

The software-to-be can be described in terms of its components (cf., UML Component Diagrams [Fowl04]) using GRL, with the same modelling elements that were used to describe stakeholders in

---

39 Feature Oriented Domain Analysis [KCHN90] (as in Software Product Line Engineering [PoBL05]).
40 An explicit FODA model is not necessary for the approach (but rather a mental model), since an explicit model of the software features is created in Stage 3/3 using GRL.
the first activity (5.1) —e.g., the ‘Screen System Library Manager’ agent ‘is part of’ the ‘Screen System’. Creating this breakdown of software components would be useful for analysis a software component’s benefit (e.g., by aggregating the benefits of the requirements allocated to the respective agent), as well as for filtering, visualising, and managing the resulting model.

Of the choice between GRL’s intentional elements Task, (Hard)goal, or Softgoal for modelling a software agent’s functional capabilities, the definitions of the former two in [Liu12] seem appropriate:

i. “A task specifies a particular way of doing something ...[and] can also be seen as the solutions in the target system ... provid[ing] operations, processes, data representations, [etc.]”.

ii. A “system goal [typeOf hardgoal] ... generally describes the functional requirements of the target system”.

At first glance, it would seem that a ‘(Hard)goal’ element is the most suitable for representing a system feature (as a functional requirement). However, arguments for using ‘Task’ elements are that:

a. System features are particular ways [Yu95, p.14] of treating stakeholder problems;

b. System features exist in the solution domain (as established in Chapter 2.1.9);

c. System features as ‘functional requirements’ are ‘operational requirements’ [Davi93] (cf., i);

d. Software components (agents) are incapable of desiring goals, but they can perform tasks;

e. i*’s documentation proposes that leaf level tasks can be translated into ‘leaf-level system requirements’ [Hork10, fig.4.7.3];

f. If the same element (goal) is used to represent system features and expected benefits, then there will be no visual distinction between them, reducing the model’s comprehensibility.

In terms of GRL’s semantics, the choice of intentional element seems trivial. Indeed, GRL’s official documentation provides an example of a system requirement represented as either a (Hard)goal “Voice Be Transmitted”, or a Task “Transmit Voice” [Liu12, sec.4.2.3]. In this example, the distinction between the two seems purely representational (i.e., phrased as a ‘state of the world’ or an ‘activity’).

Hence, the pragmatic argument (f) for choosing ‘Task’ elements would appear to hold the most currency. As a final note on the rationale for modelling software features as tasks, Yu provides the guidance that “the means [in a means-end link] is usually expressed in the form of a task” [Yu95, p.31]. However, this is not fully applicable to our approach, since in chains of means and ends links (which are not intended to be modelled with i* or GRL), a means is an end in another context. Hence according to Yu’s guidance, all elements in the chain would be tasks, except for the root. An analogy highlighting this inapplicability, is claiming that surfaces above one’s head in a building are usually ceilings, and those below one’s feet are usually floors. In the context of i* and GRL, all buildings are bungalows, but in the context of chains of instrumental value, most buildings are skyscrapers.

Figure 5.12 shows a GRL diagram representing the software agent-to-be’s proposed features (i.e., tasks they will be causally responsible for), derived from the Feature Model in Figure 5.11. (To enrich the description of functional refinement, the decomposition links could be annotated along the archetypal dimensions of optionality, multiplicity, temporality, synchronicity, and so on, as are defined by ConcurTaskTree’s task-decomposition meta-model [PaMM97]).

---

41 This is similar to KAOS’s guideline that goals should be distinguished from their ‘operationalizations’ by specifying goals in the past tense [Lams09a, p.317].
The tasks that the software agent is modelled to be capable of in Figure 5.12 relate to many use cases and classes, because they operate on the abstract class of ‘Media’ (including all conceivably supported types of ‘Media’ to be used in the digital signage system—as shown in Figure 5.9’s UML Class diagram). The modeller has a choice as to whether or not to include one task in the GRL model (agent capability) for every use case (i.e., a separate ‘Delete’ capability for every subtype of ‘Media’). Overall, if the specialised software capabilities (such as ‘delete advert’ rather than ‘delete media’) would require significant extra development work, or if one or more particular specialisations would be significantly more beneficial than the others, then it is worth modelling each specific capability. Furthermore, from a re-use perspective, the more specialisations modelled, the higher the recall rate for similar benefits in future projects. However, as a consequence the model would become larger and so less likely to be read by a non-committed stakeholder (and a similar recall level could instead be achieved with ontologies of the domain that provide the information that an ‘Advert’ is a type of ‘Media’).

**5.3.2 Modelling Non-Functional Requirements (Qualities)**

Significant benefits are often realised not by introducing new functionality into the problem domain, but rather by introducing better quality functionality than currently exists. Even where this is not the case, non-functional requirements play a crucial role in determining the benefit that would be realised by the implementation and operation of functional requirements. For example, if a user were asked to describe the advantage of a printer over pen and paper transcription, then the ‘print’ functionality would be imagined to have certain qualities considered typical for the imagined printer (e.g., legibility of print quality perhaps measured by dots-per-inch, signal-to-noise ratio, etc.).

**Stage 1/2: Defining Non-Functional System Requirements (Properties/Characteristics)**

The output of this task is a set of non-functional requirements that enrich the set of functional requirements that were previously defined. They should be related to leaf level features (e.g., [LEAF2] of Figure 5.11), or to higher-level features on the condition that each of its lower-level features are equally constrained by it; Generally, root level features (e.g., [ROOT1] of Figure 5.11) should only
be related to non-functional requirements concerning the architecture of the software [DKPW07, p.6], i.e., for specifying ‘System qualities’ rather than ‘Capability Qualities’ [SBJA98].

In order to guide elicitation of a feature’s non-functional requirements, numerous taxonomies of software qualities (e.g., usability, performance, security, etc.) exist, of which Hewlett-Packard’s FURPS [GrCa87] and ISO 9126-1 [Iso01] are the most popular. Non-functional requirements could be derived from these qualities for a feature, based on their pertinence to the benefit expected from a feature. For example, the requirements engineer might examine [LEAF4] of Figure 5.11, and upon looking at FURPS’s ‘Performance’ qualities, realise that ‘the frequency of the news feed interval updates’ would influence the benefit that could be realised by it ([LEAF4]).

Guidance on representing non-functional system/software requirements for this approach is required, since the way in which they are represented affects the way in which their benefits would be modelled. For example, a ‘security’ concern could be represented as the following (where ✘ indicates that they are not considered suitable for the approach, and ✔ means that they are):

a. ✘ ‘be secure’ (a generic system-wide software quality, belonging to ‘F’ in FURPS);
b. ✘ ‘probability of data theft<0.1’ (a measurable ‘acceptance test’, cf., Chapter 4.3.1);
c. ✘ ‘security features should take ≤2 weeks to develop’ (a requirement on project scheduling);
d. ✘ ‘use encryption’ (a function proposed in order to satisfy the former three);
e. ✔ ‘encrypted with ≥256-bit key length’ (a property/constraint/characteristic of a software function in the system-to-be, whose satisfaction can be guaranteed solely by the software);
f. ✔ ‘users should change passwords once every 30 days’ (a property/constraint/characteristic expected from a human agent in the system-to-be);

(✠) Invalid Non-Functional Concerns to Model in this Approach (a, b, c, d):

Generic qualities (a) should not be defined for their relation to expected benefits due to their ambiguity, and so should be refined into one of the latter more specific types.

Acceptance tests (b) that not are not something that the software agent can be directly and solely causally responsible for, but which represent an advantageous change in the state of the world for a stakeholder, should be represented as expected benefits (Section 5.2.1), rather than as requirements that the system or the software agent must be able to control.

Similarly, project requirements (c) that the system-to-be-built cannot not responsible for (instead, belonging to the people building the system) are not of interest, but could be represented as standard goals/softgoals in GRL — rather than an expected benefit or non-functional system/software requirement. (Project requirements, e.g., on time/cost, may be necessary for the software to be pertinent to the problem context, but they are not benefits of the software; They exist only as constraints on the introduction of a solution, i.e., software agent(s). Nevertheless, software requirements can positively/negatively influence them, and benefits can depend on them.)

---


43 The difference between non-functionality and functionality lies in representation rather than concern [Glin07], and while a software requirement is a system requirement, the converse is not true [Lams09a].
Finally, functional ways of achieving qualities (d) should be treated as functionality derived from a quality (Section 5.3.1).

(✓) Valid Non-Functional Concerns to Model in this Approach (e, f):

In the context of this framework, when the benefits of non-functional requirements (NFRs) are in question, it is the properties, constraints, or characteristics of functionality to be caused by a component of the system-to-be that are to be examined. These NFRs should primarily refer to world phenomena as software input/output (rather than problem world phenomena [ZaJa97]), as in (e). The NFRs may also refer to phenomena to be caused by non-software agents within the system (e.g., expectations on humans), as in (f), in order to address non-functional system requirements.

Guidance on the textual construction and definition of non-functional requirements is equivalent to that provided by Section 5.3.1 for the purpose of this approach, with regards to precise ‘Fit Criterion’ being the essential aspect. The primary exception is that the specific quality type from FURPS should be recorded as a field of the requirement. This is especially important where functionality was originally proposed as non-functional, e.g., ‘use encryption’ (d), since it enables:

- completeness analysis of a feature’s required qualities, to identify missing requirements;
- conflict detection & resolution among a feature’s qualities, enabling re-use of generic interactions such as that security often conflicts with (and is often preferable to) usability (Chapter 4.5.1).

Stage 2/2: Model Non-Functional Requirements as System Agent Capabilities

Whereas GRL would represent non-functional requirements as softgoals, this approach models them as tasks owned by capable & causally responsible actors in the system (software or human), since:

- All non-functional qualities $Q$ of functionality $F1$ are eventually operationalised by other functionality $F2$ or architectural configuration $A$ [Paec04]$^{44}$, and so $(F2)$ or $(A)$ can be considered ‘particular courses of action’ for implementing $(F1)$ to quality $(Q)$. Accordingly, a GRL task “specified as a subcomponent of a higher task restricts the higher task to that particular course of action” [Yu95, p.33]. For example, ‘remember login for 2 weeks’ $(F2)$ restricts ‘remember login’ $(F1)$, contributing to the ‘Usability’ $(Q)$ of $(F1)$;
- An ontological analysis of i* & GRL’s softgoal concept concluded that “a softgoal can only be evaluated subjectively”, precluding “explicit measures” [MoHM10] that are necessary to be able to unambiguously include non-functional requirements in causal propositions. In summary, one “should not confuse soft goals with non-functional goals” [Lams09a, p.271];
- ‘How fast’ a software feature is, for example, concerns how the software is to be built, which, in the context of this thesis, concerns the solution to realising stakeholder benefits—and as aforementioned in Section 5.3.1, it is useful to have a (visual) distinction between the two.

Furthermore, whereas in Chung’s NFR approach (Chapter 4.5.3), non-functional (and implied intrinsically valuable$^{45}$) qualities such as Usability might be contributed to by a function, in this approach, a non-functional requirement is considered as a necessary (and often implicit) property of a function. For example, in Figure 5.13, the requirements engineer examined the FURPS quality list for software feature [A], recognised that the Speed quality would influence the benefit that would be realised by [A], and hence proposed non-functional requirement [B]. While the actual non-functional

$^{44}$ See also [Lams09a, p.271] for a discussion of ‘non-functional’ being a ‘category’ of behavioural goals.

$^{45}$ In the ‘closed world’ ontological view [CWAM75] (i.e., NF’s aren’t modelled as instrumentally valuable).
property value (60 seconds) might at this stage be entirely based on ‘gut feeling’, later analysis of its
contribution to its expected benefits will either provide a reasoned argument for it, or result in a different
value. Finally, at this point it becomes useful to distinguish non-functional requirements in the GRL
diagrams, as is done with the element label prefix of ‘\{F\}’ for functional and ‘\{NF\}’ for non-functional.

Finally, the requirements engineer should be warned that omitting non-functional requirements
leads to causal propositions about the beneficial effects of a software feature having a higher than
otherwise composition of ‘Unknown Component Causes’ (Chapter 2.1.2.1), which increases model
uncertainty [OBDE06], and hence reduces the potential for incorrect assumptions to be corrected.

Figure 5.13: GRL model of a Non-Functional Software Requirement \{NF\} as a decomposition of a
Functional Software Requirement \{F\} (constraining \[A\] rather than \[B\] being a distinct task)

Verification against the Required Capabilities for the Approach (Chapter 4)
The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

- ✔ [A3] Requirement- Decomposition: “Be able to refine and decompose a requirement into atomic concerns that
can be assigned and enforced by one agent in the system.”
- ✔ [A5] Non-Functional Requirements: “Provide guidance on representing quality (non-functional) requirements,
i.e., with various degrees of satisfaction.”

5.4 Construct & Refine Goal Graph

This activity takes as input the sets of Stakeholders, Isolated Expected Benefits, Software Features,
and Software Qualities, previously populated by the preceding activities (Sections 5.1 to 5.3) to a high
degree of individual-element-completeness, rather than set-completeness. The primary output of this
activity is a GRL model connecting the above sets through numerous levels of actor-boundary-
intersecting goal contribution or decomposition links, where each link represents a link in a chain of
instrumental value derivation from something ‘more’ intrinsically valuable. The construction of
these links will inevitably trigger the creation of new elements in the aforementioned input sets,
causing re-instantiation of the previous activities, and increasing the completeness of the input set.

5.4.1 Establish Initial Links between GRL Elements

The premise of this activity is that every modelled capability of the software-to-be agent (Section
5.3) causally contributes to the achievement of some modelled expected benefits (Section 5.2) for
some modelled stakeholders (5.1). This activity is concerned with making these links explicit to be
later explored and enriched.
Stage 1/2: Choosing GRL Link Elements for Modelling the Causal Propositions

According to the aforementioned sets of elements-to-be-linked in this approach (in Section 5.4), the modeller requires element-element links for describing the relationships between a—i in Table 5.3:

- **a. Software Agents (Actors) → responsible for → Software Capabilities (Tasks);**
- **b. Stakeholders (group S2 in 5.1) (Actors) → responsible for → System Capabilities (Tasks);**
- **c. Stakeholders (group S1 in 5.1) (Actors) → desire → Expected Benefits (Goals);**
- **d. Software Capabilities (Tasks) → necessary/sufficient for → Software Capabilities (Tasks);**
- **e. Software Capabilities (Tasks) → necessary/sufficient for → System Capabilities (Tasks);**
- **f. Software Capabilities (Tasks) → necessary/sufficient for → Expected Benefits (Goals);**
- **g. System Capabilities (Tasks) → necessary/sufficient for → System Capabilities (Tasks);**
- **h. System Capabilities (Tasks) → necessary/sufficient for → Expected Benefits (Goals);**
- **i. Expected Benefits (Goals) → necessary/sufficient for → Expected Benefits (Goals);**

(This section focuses on the links required for causal propositions, i.e., (d—i)). Links (a, b, c) have been previously demonstrated in Figure 5.13 for a & b, and Figure 5.8 for c. Links between actors have already been demonstrated in Figure 5.14.)

Table 5.3: Required relationships between elements for causal propositions on value creation

<table>
<thead>
<tr>
<th></th>
<th>Software Agents</th>
<th>Stakeholders</th>
<th>Software Capabilities</th>
<th>System Capabilities</th>
<th>Expected Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Agents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Capabilities</td>
<td></td>
<td></td>
<td>(a)</td>
<td></td>
<td>(d)</td>
</tr>
<tr>
<td>System Capabilities</td>
<td></td>
<td></td>
<td>(b)</td>
<td>(e)</td>
<td>(g)</td>
</tr>
<tr>
<td>Expected Benefits</td>
<td></td>
<td></td>
<td>(c)</td>
<td>(f)</td>
<td>(h)</td>
</tr>
</tbody>
</table>

Links (a, b, c) are between actors and intentional elements, and so the viable GRL links are *Actor Boundary*, and *Dependency*. Links (d—i) are between intentional elements, and so the viable GRL links are *Decomposition, Means-end*, and *Contribution*. The remainder of this subsection will discuss the applicability and usage of these GRL links within this thesis’ framework.

For links (a, b, c), GRL’s ‘actor boundary’ element is the most suitable since it represents ‘ownership’ of intentional elements, in that “placing elements inside the actor boundary suggests that they form part of that actor’s rationale” [MoHM10] for their desires/dependencies/actions in the system. However, pragmatically speaking, a dependency link could be made from the various actors to the same intentional element, which explicitly indicates their “desire” for it [Yu95, p.39].

As argued in Chapter 2.1.2, describing that a software feature will contribute or produce an expected benefit is to describe a causal proposition, which should be described probabilistically and counterfactually in terms of the software capabilities’ degree of sufficiency or necessity for the benefit. Fortunately, due to GRL being primarily a language for describing design rationale, it already has basic constructs for describing an intentional element’s necessity or sufficiency for another. (‘Basic’ in that they are not underpinned by a particular theory of causation (discussed in Chapter 2.1.2.1)), it is not clarified what it means, for example, for an element to be necessary to another (e.g., necessary in the project, or necessary in all projects), or which world state an effect is relative to, i.e.,
counterfactual causes.) Hence for links (d — i), the choice between the links is guided by whether the first element is either necessary (Decomposition), fully sufficient, optional or not necessary (Means-End), or would be at least somewhat sufficient (Contribution) for the second element.

Usage of the previously discussed Contribution link in this framework deviates from the official GRL ontology, which defines that a contribution link “describes how one intentional element contributes to the satisficing of another intentional element”, where the latter intentional element must be a softgoal [Liu12]. Consequently, GRL’s qualitative contribution labels (Chapter 4.5.2) are not used. Finally, given the above deviation, a Contribution link could be described with an extent of contribution that is fully sufficient for the destination goal, and hence would be semantically similar to a Means-End link. However, Means-End links play an important role in visually distinguishing optionality (although OR annotation on a contribution link could achieve the same).

The next section will focus on links (d — i) required to specify causal propositions for value creation.

Stage 2/2: Constructing Initial Causal Links between Software Capabilities & Benefits

Depending on whether the software capabilities were elicited ‘solution first’ or ‘problem first’, it would be easier to either perform ‘bottom-up’ goal abstraction (asking ‘why?’) from the Software Features/Qualities, or ‘top-down’ goal refinement (asking ‘how?’) from the Isolated Expected Benefits, respectively. In the former case (‘bottom-up’), the obvious first step for establishing links is to examine the stakeholder roles who would use the proposed software features for ‘direct’ benefits (i.e., stakeholder groups S1.a and S2.a). In the latter case (‘top-down’), the software features could be examined for their influence on the objects they manipulate, in order to match up with expected benefits via their ‘Object’ field.

As an example of the former (‘bottom-up’) case, the software feature ‘Delete Media Automatically’ ([A] in Figure 5.14) could be evaluated for its contribution to the expected benefits owned by the same actor who wants to use it (e.g., as defined by the UML Use Case Diagram in Figure 5.9). If no such Expected Benefits are modelled to exist yet, the modeller may elicit them ‘just in time’. As a result, the contribution link [L1] shown in Figure 5.14 might be proposed, having a positive (+) causal influence on its realisation (contributing 100% of [C]’s Target of a 99% reduction), but not being ‘necessary’ for the benefit because other ways of achieving the Target exist (discussed on ‘necessary causes’ in Chapter 2.1.2.1).

Finally, whenever a contribution link is created from an element that has siblings related via Means-End links (e.g., [A]’s alternative ‘via UI (Manual Deletion)’), then an implicit contribution link exists in the underlying model between those siblings and the element contributed toward (e.g., [C]). These links are necessary in the model in order to be able to compare and evaluate alternatives. However, drawing these links would quickly obfuscate a diagram (especially where there are multiple elements contributed toward), and so they can be toggled on and off in the framework’s software tool.
5.4.2 Describing Contribution Mechanistically with Numerous Contribution Links

Initially, links between the intentional elements (requirements and benefits) may not describe causal propositions sufficiently mechanistically (as discussed in Chapter 2.1.2.3), since the priority in the first instance is to model something to be later improved, rather than to be overly-occupied about correctness and end up modelling nothing. However, throughout this thesis, we have made reference to ‘benefits’ being advantageous outcomes of objectives, and to instrumental value being derived from something else. Conversely, up until this stage of the approach, we have made reference to ‘isolated’ expected benefits, and to ‘initial’ links between intentional elements.

Hence, this activity is concerned with expanding a chain of instrumental value currently consisting of two intentional elements (as in contribution link [L1] in Figure 5.14), to a chain consisting of many more elements. In doing so, the software feature’s benefits and instrumental value are explored, whilst concurrently describing the expected benefit’s causal mechanism(s). With reference to Figure 5.14, the chain of instrumental value description can be expanded between contribution link [L1], or by constructing and expanding above contribution link [Ln…]. At which of these points the expansion first occurs, depends on whether the causal mechanism of [L1] is sufficiently transparent, or in simpler terms, whether asking ‘how does [A] cause [C]?’ would lead to the elucidation of non-modelled software behaviours or expected benefits. A practical technique for performing such analysis is to decompose the cause/effect variables to examine how they are determined or composed. For example, in the causal proposition that “driving faster causes fuel consumption to increase”[FoGe83, p.8], it is possible to expand on the composition of those variables to describe the causal proposition more mechanistically; e.g., driving faster → throttle position → flow of

An introduction to asking ‘How?, Why?, & How Else?’ questions about goals exists in [Lams09a, p.311], but it does not go much beyond that those questions should be asked, rather than how to ask them.
fuel &→ engine speed &→ fuel consumption. Since knowledge of the problem’s causal mechanism is required, both stakeholder groups S1 (problem owners & sufferers) and S3 (experts) should be consulted (Section 5.1).

5.4.2.1 Adding Contribution Links between Existing Contribution Links (Expanding)

While contribution [L1] in Figure 5.14 seems complete in that it is self-explanatory (compared to other possible higher-level effects of [A] e.g., ‘Increased[Job Motivation]’), there exists a vital and as-yet implicit composition of the ‘Time Taken to Remove Expired Adverts’ [C] variable. Consequently, if two domain novices were set the task of debating whether [A] should be implemented given the information in Figure 5.14, then the proponent’s argument would be based only on their general knowledge that ‘Automation’ [A] tends to reduce the time taken to do things. As such, the opponent might struggle to understand the pertinence of contribution link [L1], since they might not believe that deleting a digital file takes very long.

Hence, in order to make the proponent’s argument more informed and transparent (leading to the detection of errors in belief or reasoning), the composition of [C]’s variable should be refined, leading to the creation of new Expected Benefit elements. For example, the ‘Time Taken to Remove Expired Adverts’ [C] depends on the time between an advert expiring and the removal process starting (t1), and then the time taken after t1 to get to a computer (t2), to log into the screen system (t3), to search for the expired media item (t4), and finally to delete the media item (t5). The crucial aspect is that the first sub-variable (t1) consumes the vast majority of [C]’s time, as opposed to the latter four (t2–t5). Therefore, in Figure 5.15, elements [C1] and [C2] are decomposed from [C] so that their non-equal contribution to [C] can be specified. Given that [C1] and [C2] completely compose [C] in the model (in the sense that [C]’s Magnitude values are the sum of [C1] and [C2]’s), a ‘Decomposition & Contribution’ link is used. (In GRL, this link is referred to an ‘AND Contribution’ and stipulates necessity [Liu12], but in this framework its significance is upgraded to signify ‘part→whole’ composition — as the ‘decomposition’ name generally implies, and as UML implies for its ‘composition’ links [Fowl04]).

As a brief walkthrough of the numerical labels in Figure 5.15, [C] is judged to typically take 7 hours 10 minutes (430 minutes), split between 7 hours (420 minutes) to perceive that an advert has expired [C1], and then 10 minutes to actually perform the removal [C2]. The requirements engineer and the stakeholders then decided that:

- For [C1], the 420 minutes currently taken to perceive that an advert has expired should be reduced by at least 419 minutes, so that it would take no longer than 1 minute to start the removal process, hence [C1]’s 99% (419/420=99.8%) reduction ‘Target’ label.
- For [C2], the 10 minutes currently spent performing the removal of an expired advert should be reduced by 10 minutes (i.e., to instantly), hence [C2]’s 100% reduction ‘Target’ label.

Owing to links [L1.3] and [L1.4] being of the ‘Decomposition & Contribution’ type (and hence the availability of attribute inheritance and calculation in the framework’s software tool), then [C]’s Magnitude values (e.g., its ‘99%’ Target) can be calculated according to a non-weighted summation aggregation equation, e.g., 

\[
[C].Target = ([C1].Target + [C2].Target) / ([C1].AsIs + [C2].AsIs) = 429/430 = 99.8\%.
\]

Finally, the contribution extent labels (on [L1.3] & [L1.4]) are calculated using the contribution

---

47 E.g., the ‘Advert Dissemination’ role player might only notice an expired advert after walking past it.

48 It is useful to consider the four types of time lag: Perception, Evaluation, Action, Inertial Response, with the three intersecting decisions: To Take Cognizance, On What To Do, and To Start Intervention [Wein63].

49 The consequences of ‘overloading’ GRL’s ‘decomposition’ visual notation in this case has no significant negative implications, since generally a part is necessary for the existence of a whole.
‘if-then’ rules later described in Section 5.5.1.2, but for Figure 5.15’s simple example, [L1.3]’s (98%) was calculated on the assumption that [C1] would likely be fully satisfied and that [C1] and [C]’s scale are linearly correlated (coefficient of 1), and hence: \((L1.3).\text{ContributionLabel} = (C1).\text{Target} / (C).\text{Target} = (419/429) = 98\%\). (Note that the above description of contribution serves only to explain Figure 5.15. A systematic guide is located in Section 5.5.1.2).

5.4.2.2 Adding Contribution Links above Existing Contribution Links (Abstracting)

Following the adequate expansion of the initial contribution link between the two intentional elements, the activity becomes oriented around explaining ‘what benefit is expected to come from an expected benefit?’ (or in other words, ‘what other problems would it influence?’), by constructing contribution links after the first expected benefit. By doing so, chains of instrumental value from a software capability are modelled, with each additional contribution link approaching the source of ‘intrinsic value’ from which all other value in the chain is derived. Hence, the modeller requires an understanding of instrumental value (especially ‘Common Misconceptions about Instrumental Value’ in Chapter 2.1.1.3), mechanistic causal propositions (especially the guidance on ‘Common Misconceptions about Causation’ in Chapter 2.1.2.4), and indirect benefits (Chapter 2.1.3.1).

In theory, if all of the expected benefits had been elicited at activity 5.2, and all of the software features had been derived at activity 5.3, then this this activity should be a simple ‘connecting the dots’ exercise. However, in practice, neither of those two sets will be complete, and so this activity will prompt the elicitation of either expected benefits or software features, i.e., drawing new ‘dots’. Just as the activity of ‘eliciting stakeholders who would benefit from the software’ (5.1.2) requires boundary conditions to prevent unmanageable size, guidance is also required in this activity. Hence, two questions are proposed in Figure 5.16 to determine if an expected benefit should be ‘abstracted’ higher, or if it could be considered sufficiently intrinsically valuable.

Criteria for determining if an Expected Benefit should be further ‘abstracted’ higher

1. Is the proposed abstracted benefit more than logically coherent: is it plausible and reasonable in the context?
2. In a universe where only the already modelled benefits could occur, would the owner of the newly proposed abstracted benefit be in a better position due to it?

Figure 5.16: Criteria for stopping a chain of instrumental value exploration from a requirement.
The first question avoids modelling ‘what could be the benefits?’, such as modelling that a ‘Write Text’ feature of word processing software would reduce the time taken for ‘Joe’ to write a bestselling novel — if ‘Joe’ is actually a technical writer having no such intentions. This does not mean to say that unlikely benefits should not be modelled (indeed, confidence and credibility are later described), but benefits should be rational and fit the context of the problem domain.

The second question avoids modelling syntactically different, but semantically equivalent benefits, since the state of the world should be perceptibly and not insignificantly better for the owner stakeholder. Additionally, the question reinforces that while one stakeholder might be able to ‘connect the dots’ (and see others where they are not drawn), another might not. (The ‘connect the dots’ analogy refers to the ability to understand that ‘dots’ [A], [C], [D], [E], [F] in Figure 5.18 are causally related. The IS community similarly refers to ‘line of sight’ [SiWo09].) Indeed, due to the compartmentalised (often termed ‘stovepipe’) structure of organisations, one person rarely has visibility of all aspects of a problem to be solved [Geor79], and so a holistic view should not be taken for granted. As a final consequence of this question, proposed benefits that purely translate the units of an existing benefit, should not be modelled as a new benefit with the same object & focus. For example, a ‘reduction of product X’s weight’ specified in grams, should not be modelled to contribute to a ‘reduction of product X’s weight’ specified in ounces, otherwise its calculated weight would be doubly reduced. Translation of units can instead be achieved using this framework’s software tool (using ‘GNU Units’ [Mari14], or ontologies for specialised domains, e.g., [DoLo05]).

Finally, this activity significantly depends on the output of the ‘Elicit Stakeholders’ activity (5.1), since ‘what is good about benefit X?’ actually means ‘what is good about benefit X for the system’s stakeholders?’; There can be many ‘final’ sources of value (Chapter 2.1.1.2) for a software feature, each owned by different stakeholders or organisations. An obvious starting point is to examine how the organisations that the benefit owner ‘is-part-of’ (GRL link) would benefit, e.g., the ‘Marketing Department’ in the case of the ‘Advert Dissemination’ role (Figure 5.4).

In organisations, various horizontal and vertical Business Units (BUs) (in a matrix organisation structure [Kroo95, p.253]) might be beneficiaries of a software feature. In such cases, modelling how the realisation of a benefit for one BU achieves some benefit to be realised for another, is similar to the process of modelling organisational strategic alignment [WaGh06] (but at a tactical/operational level due to the focus on software features). As opposed to the majority of strategic alignment modelling approaches which consider ‘alignment’ as binary (i.e., aligned or not) [WaGh06, p.4], or as ‘box ticking’ and management-pleasing activities (Figure 5.17), this approach supports a ‘degree of alignment’ (via estimated contribution to a benefit’s scale). Hence, vague and useless interpretations of alignment (Figure 5.17) would appear inadequate when modelled with this approach, due to the benefits not being rooted in observable and measureable problem domain phenomena that embrace, rather than ignore uncertainty.

Figure 5.18 exemplifies the output of this activity, in that contribution links above an Expected Benefit now exist (and accordingly, expected benefits [D], [E], and [F] are added), showing how the software feature [A] is instrumentally valuable.
Figure 5.17: In practice, ‘consistency’ (i.e., ‘alignment’) of a software project with organisational objectives is often claimed without practicality, and as Boolean [WaGh06, p.4]. (© [Adam07b])

Figure 5.18: Adding contribution links above [C] in order to construct a chain of instrumental value (as opposed to adding links between [A] and [C], as was previously discussed in 5.4.2.1).
(Figure 5.18 deviates from standard GRL diagrams in numerous ways (explained by Chapter 4’s feature comparison), but most importantly, standard GRL diagrams do not permit hardgoals as contribution link destinations, and links other than dependencies are not permitted between actor boundaries [LMER11].)

If the modeller’s time is limited, a useful heuristic observed by Ohno (in the development of Lean’s 5 Why’s? technique), is that the root cause of a problem can often be found after asking ‘Why?’ five times [Ohno88], and so a ‘good enough’ exploration of a software features value would likely be achieved when a chain of five expected benefits exists. The primary pragmatic concerns are that the ‘benefit’ side to a feature’s value equation is adequately represented (this tends to be less of a problem for costs [Boeh03]), that scope for planning ‘necessary non-software intervention’ to realise expected benefits is identified, and that risks to value realisation are identified. For example, Figure 5.18’s [F] shows that the ‘Marketing Department’ might not consider the software project to be successful unless the productivity of their ‘Graphic Designer’ position is increased.

5.4.3 Describing Assumptions on Variables Expected to Hold True

Finally, numerous ‘Expected Benefit’s will be specified such that they depend on variables currently external to the model holding true. For example, [D]’s scale and target is defined in absolute ‘hours per month of menial work’ in Figure 5.10. While the achievement of that target primarily depends on the contribution of the initiatives — i.e., the new software capabilities, it also depends on assumptions about other variables holding true. In [D]’s case, contribution link [L2] toward it depends on the number of adverts that expire every month, and hence that need removing. This numerical assumption could be modelled with a GRL Belief element that is linked to the vulnerable intentional element using a decomposition link to show its necessity. (Where there is uncertainty in this assumption, it could be useful to describe it as a random variable, e.g., that it follows a normal distribution with a certain mean and standard deviation.)

![Figure 5.19: Describing an assumption expected to hold true for expected benefit [D] to be realisable.](image-url)
Alternatively, given that [C2]’s ‘Scale’ is an un-aggregated and non-summarised Flow, (see Section 5.2.2), i.e., ‘per removal of an advert’, the contribution link [L2] could be split to include a new Expected Benefit inserted in between, to represent a reduction in the total time per month taken to remove expired adverts (i.e., a statistical summary that takes the frequency of ‘advert removals’ into account). [Assumption 1] in Figure 5.19 would then be built into the new Expected Benefit, and so the GRL belief element would serve mostly as a visual highlight of the built-in assumption. This has the advantage of achieving a more mechanistic description of the contribution link [L2], but the disadvantage of cluttering the diagram and consuming more of the modeller’s time. In conclusion, such assumptions should be built into the Scales of the Expected Benefits (since assumptions are essentially variables in an equation determining the Expected Benefit’s Scale value), and GRL’s belief elements should be used to draw the diagram viewer’s attention to the most risky variables.

Finally, there are times when an Expected Benefit is threatened by an obstacle. Since GRL has no notion of such a concept, we include KAOS’s, which defines it as “An obstacle to an assertion is a pre-condition for the non-satisfaction of the assertion. The obstructed assertion is in general a goal.” [Lams09a, p.336]. It is especially important that obstacles (in addition to other interventions necessary for the Expected Benefits) near the top of a chain of instrumental value derivation are elicited and risk assessed, given that the software project team is likely to pay more attention to risks lower down the chain toward the software capability. For example, the obstacle in Figure 5.20 threatens the realisation of [F], and hence the value of software capability [A].

![Figure 5.20: Describing an obstacle to [F]’s realisation.](image)

When evaluating obstacles, it is important to understand its owner (i.e., who is most affected and most responsible for assuring its mitigation), e.g., the ‘Marketing Department’ actor, as well as any other actors it involves, e.g., the ‘Graphic Designer’ role; Obstacles on human behaviour can be better assessed by considering the agents associated with the role in order to understand their personal interests, and hence the likelihood of the obstacle applying to the particular project context. Furthermore, retrospective modelling of obstacles (i.e., after the software capability has been implemented) enables the codification and hence future re-use of valuable ‘lessons learned’.
5.5 Estimate Predicted Contribution

At this stage of the approach, a network exists of software features contributing to expected benefits, contributing to other expected benefits, and so on — as exemplified by Figure 5.18. While the modeller has so far been guided in the prescriptive activity of specifying targets for Expected Benefits, there has been little guidance on the predictive activity of estimating contribution between the software requirements and benefits. This is important, since describing how beneficial a feature would be is essential to describing what the benefits of a feature would be. As a crude example, clearly the benefit of a £10 cost saving is different to a £1000 saving, and if the former is desired and the latter is predicted, then more or better intervention is required.

Hence, this activity elicits and models stakeholder belief about the extent of the ‘some contribution’ between the intentional elements. All stakeholder groups from Section 5.1 should be involved in this activity, especially the domain experts (those most qualified to estimate degrees of contribution from software features to their associated benefits). The following subsections incrementally build upon the metadata that describes the causal proposition underpinning each contribution link.

5.5.1 Describing Contribution Metadata based on an ‘If-Then’ Causal Proposition

The contribution link [L1.1] in Figure 5.18 represents the causal proposition that if the software feature [A] (along with its necessary components such as non-functional requirements on it) were implemented and used, then it would cause a decrease in the variable described by [C1]’s ‘Scale’ field. The subsequent sections describe the metadata that build up this ‘if(then’ causal proposition.

5.5.1.1 Describe the Polarity of the Contribution’s Causal Proposition

The first and most primitive dimension of the causal proposition is its Polarity. While some researchers have argued that Directionality between causal variables should be described instead (i.e., as one variable increases, the other increases in the Same or Opposite direction), this can lead to faulty conclusions due to the existence of ‘stocks and flows’\(^\text{50}\). Hence, any contribution between two intentional elements (between software capabilities and/or expected benefits) should be described in terms of being either Positive or Negative (as in causal loop diagrams [Ster00, p.141]):

A positive causal link between two intentional elements should be interpreted as that:

If the cause increases, then the effect would increase above what it would otherwise have been;

If the cause decreases, then the effect would decrease below what it would otherwise have been.

Whereas, a negative causal link should be interpreted as that:

If the cause increases, then the effect would decrease below what it would otherwise have been;

If the cause decreases, then the effect would increase above what it would otherwise have been.

---

\(^{50}\) For example, it would be wrong to conclude that a decrease in a population’s birth rate directly subtracts from its population (rather, deaths do), given that an increase in births increases the population [Rich97].
For example, contribution link [L1.1] between [A] → [C1] can be interpreted as:

\[
\text{If the causal variable Boolean: IsFeatureImplementedAndUsed(Delete Media via Schedule) [A] increases from false (as-is) to true (to-be), then the Number: Reduction of Time Taken to Perceive the Need to Remove an Expired Advert [C1] (effect) would increase above what it would otherwise have been.}
\]

The final (italicised) parts of the four ‘if-then’ rules incorporate the ceteris paribus assumption, i.e., ‘all else being equal [unchanging]’, which makes it possible to practically model world phenomena. Ceteris paribus is “the most commonly used assumption in economics” because it ‘puts the outside world [i.e., exogenous variables] on hold’ [Lind02, p.14], and so avoids the incorrect interpretation that the effect variable will always be increased if the cause variable is increased. (For example, increasing a bathtub’s tap (flow) would increase the water level, but not if the tub’s plug is removed in the intervening time. Similarly, if software requirements are prescribed to reduce the time consumption of a business process via automation, but the demand for the output of the process falls, then the total cost of the process after the software’s implementation may be increased, rather than reduced as intended.) Another opportunity for eliciting risks to benefit realisation is presented at this point, via making explicit any assumptions on variables that are expected to remain equal, as is exemplified by Figure 5.19.

Sometimes, a contribution link’s polarity may seem to be both positive and negative (or, there is a monotonic contribution), perhaps depending on the extent of the cause being implemented. In such cases, the expected benefit is likely conflated in that it contains both a desirable and an undesirable state of the world. For example, contribution link [L4] in Figure 5.18 could be positive and negative, since up to a point, reducing menial (i.e., non-creative) workload increases the potential for creative work, but if too much menial workload is reduced, then decision fatigue may set in, reducing the potential for creative work output. Such unanticipated and harmful effects are often referred to as ‘side effects’.

This newly identified ‘side effect’ (decision fatigue) should be modelled as its own goal (to be reduced or maintained) with an outgoing negative contribution link. This allows the former contribution link ([L4]) to be ‘purely’ positive. Consequently, reasons for the decreasing ‘marginal valuation’ of increasing reductions in menial work are made explicit, challengeable, and mitigate-able — perhaps via the modification of the software feature set, or associated expectations on human agents.

### Verification against the Required Capabilities for the Approach (Chapter 4)

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

- ✔✔ ✔✔ [B1] Contribution-Polarity: “Represent whether a software capability is believed to influence an expected benefit (or disbenefit) positively or negatively, and ideally guide representing bipolarity.”

### 5.5.1.2 Describe the Extent of the Causal Contribution

For the model to be able to answer ‘how beneficial?’ a software feature would be, the aforementioned causal propositions need to be modified to include the estimated extent of contribution. Without this, the reader of Figure 5.18 might assume that the proposed software features would be sufficient for achieving the target of each expected benefit, which is unlikely, due to:

- one expected benefit often requiring contribution from many software capabilities;
- organisation-owned expected benefits whose scope is wider than the software project;

51 Typically caused by ‘Consumer Surplus’ [MiQu07, p.24], ‘Policy Resistance’ [Ster00, p.22], or ‘Murphy’s Law’.
• aspirational (bordering idealistic) or unrealistic targets;
• alternative substitutable software features being modelled before a decision has been made about which to select, each being able to contribute different extents of satisfaction.

As such, software requirements will contribute to expected benefits with extents of satisfaction either less than, equal to, or exceeding their ‘Target’. To describe these various states of contribution, a two-stage description approach is proposed: ‘Single Point Contribution Estimation’, and then ‘Multiple Point Contribution Estimation’.

Stage 1/3: Single Point Contribution Estimation

Functional requirements have simple yes/no (i.e., Boolean) satisfaction criteria [SoVr08], and so there is one possible ‘if-then’ causal proposition to describe (in the form of ‘if the feature is implemented, then...’). This is contrary to describing the impact of non-functional requirements or other expected benefits, which have numerous (discrete or continuous) possible states of satisfaction, requiring ‘if-then; elseif-then; elseif-then; etc.’ causal propositions. This is in acknowledgement that the contribution of a non-functional requirement (the ‘then’ clause) depends on the extent of its own satisfaction (the ‘if’ clause), which itself is not guaranteed to be fully satisfied. (‘Possible states of satisfaction’ are not limited to numbers. The domain of the ‘if’ variable may be {Nominal, Partially ordered, Ordinal, Ordered Metric, Interval, or Ratio}, {Finite, or Infinite}, and {Discrete, or Continuous} [Zlat07, p.46].)

However, to increase the usability of the approach, this first stage describes only one causal proposition (‘if-then’) whose ‘if’ clause assumes full satisfaction of the cause, regardless of its type (software requirement or expected benefit)52. While this permits answers to the ‘how beneficial would this requirement be?’ question, without having to describe ‘multiple if’s’, the trade-off is that the answer is only valid when the requirement is fully satisfied.

For example, the ‘single point contribution’ causal proposition (if-then rule) for contribution link [L1.1] in Figure 5.18, is as follows (given that [C1]’s Expected Benefit specification describes an ‘As-Is’ value of 7 hours on average to realise the need for removing an ‘Expired Advert’):

If the causal variable Boolean: IsFeatureImplementedAndUsed(Delete Media via Schedule) [A]
increases from false (as-is) to true (to-be),
then the Number: Reduction of Time Taken to Perceive the Need to Remove an Expired Advert [C1]
(effect) would increase by 6 hours, 55 minutes (99%), compared to what it would otherwise have been.

Whereas statistical models of correlation for an existing dataset might define the above proposition using a (dimensionless) correlation co-efficient between the variables [A] and [C1], when eliciting uncertain quantities from people, it is considered ‘better to question people about observable quantities rather than ask direct questions about unobservable quantities, such as regression coefficients’ [KaWo98] (as similarly concluded on the guidance to use ‘concrete numbers’ in Section 5.2.2). Furthermore, whereas a traditional estimation exercise would estimate values of variables, the above proposition’s variables are reductions or increases to a variable. Among many reasons, this allows multiple inward contributions from different software features to be additively accumulated by one benefit. The above proposition for link [L1.1] can be considered as an XY data-point for, as is visualised by Figure 5.21.

52 I.e., the ‘Target’ expected benefit will be reached, or a requirement’s ‘Fit Criterion’ will be met.
Figure 5.21: Single Point Contribution from Software Feature [A] to Expected Benefit [C1] (with reference to Figure 5.18), visualised as a single data-point on an XY graph.

Table 5.4 provides a way of concisely defining the causal propositions for each contribution link, where the ‘if’ column represents the X-axis variable, and the ‘then’ column represents the Y-axis variable. The ‘if’ column is not user-editable for these ‘single point contribution estimates’, and is locked to the ‘Target’ value of the contributing element (e.g., [A], in the case of [L1.1]).

Table 5.4: Causal Propositions: Single Point Contribution Link (Table Functions)

<table>
<thead>
<tr>
<th>Link</th>
<th>IF</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>[L1.1]: [A] → [C1]</td>
<td>[True] IsFeatureImplementedAndUsed ('Delete Media via Schedule')</td>
<td>[6h 55m] [Reduction] in [Time Taken to Perceive the Need to Remove an Expired Advert] (of [7h As-Is])</td>
</tr>
<tr>
<td>[L2]: [C2] → [D]</td>
<td>[10mins] [Reduction] in [Time Taken to Perform the Removal of an Expired Advert]</td>
<td>[20mins] [Reduction] in [Daily Menial Workload] (of [3h 20m As-Is])</td>
</tr>
</tbody>
</table>

The use of a table for eliciting ‘if-then’ rules (Table 5.4) is supported by guidelines on eliciting causal relationships for systems dynamics models, as well as for eliciting parameters for regression models:

- Interactive graphics (i.e., using software to generate Figure 5.21 from Table 5.4) “seem to be particularly useful in regression problems” [OBDE06, p.151]. However, the expert should provide estimates in terms of values of Y, rather than directly in terms of Euclidean space or curve shape (and so GUI slider elements or textboxes are suitable) [OBDE06, p.148].

---

53 Each link has one row (i.e., for only X₁ as opposed to X₁...Xₙ)—as opposed to ‘multiple point contribution’.
• “The generation of a graphical representation through a succession of smaller steps ... rather than asking people to simple ‘draw the relationship’ improves knowledge elicitation by reducing the cognitive processing required of system experts in each step” [FoSt98].

Finally, the current causal propositions represented by the first three columns of Table 5.4 appear to be deterministic, i.e., certain and always-occurring. However, predictions about the future are always subject to epistemic uncertainty. Hence, the fourth column of Table 5.4 makes each causal proposition probabilistic (see Chapter 2.1.2.1), by describing confidence (likelihood/belief) in the contribution.

In the decision analysis field, representing confidence is considered essential in determining optimal decisions, especially where a choice has to be made between two options which seem to provide similar benefits [Howa07]. Making confidence about estimates indicates the risk that the modelled contributions from the software capabilities may not occur in practice.

**Stage 2/3: Describing Confidence in Predicted Extents of Contribution**

The semantics of the confidence values are that, for example (with reference to [L1.1] of Table 5.4), the modeller believes there to be a 0.95 probability of there being at least a 6 hour 55 minutes reduction (a 99% reduction in the as-is value). The ‘at least’ clause is important, since it would be almost impossible for there to be precisely a 6 hour 55 minute reduction, and any reductions higher than 6 hours and 55 minutes are considered better due to the scale’s direction of acceptability (Reduce implies that more reduction is better). Although the estimates in the ‘then’ column of Table 5.4 appear to be ‘point estimates’ (as opposed to ‘interval estimates’ or ‘probability distribution estimates’ [Kent01]), this ‘at least’ clause makes each point estimate an interval estimate, or more specifically, a Bayesian credible interval [Karl02] (since a degree of belief is attached to the true value being within the interval). For example, [L1.1] in Table 5.4 represents the range of values between 6 hours 55 minutes, and values higher than it (i.e. [6h 55m, ∞]).

Whether or not a requirement would be considered to achieve the full realisation of its expected benefits (according to the model), depends on whether the confidence-adjusted score (Expected Value [OBDE06, p.5], i.e., extent * confidence) meets the expected benefits’ ‘Target’ property.

Eliciting a single point estimate is simpler than eliciting parameters for a probability distribution. A plethora of distributions are available to statisticians, but elicitation methods for their parameters have not been proposed for the majority [OBDE06, p.221]. Even for the simplest distributions, e.g., normal or beta, it is well established that humans are unlikely to be able to reliably provide accurate estimates of variance (e.g., the standard deviation for the normal distribution) [OBDE06, p.106]. Indeed, more focused research on RE estimation (estimating the cost and risk of requirements) suggests that stakeholders find eliciting uncertainty (the width of the interval of possible values) more challenging than confidence (the degree of belief in a certain value) [Herr11]. Hence overall, “asking experts for probabilities and credible intervals is a reasonable approach” [OBDE06, p.106].

However, so far there is no reason why conservative estimates should not be made in order to maximise the confidence (probability) score. For example, an estimate for [L1.1] for ‘1 hour’ with ‘0.5’ confidence could be considered equivalent to an estimate for ‘30 minutes’ with ‘1’ confidence, at least when comparing their Expected Values (EV). Hence, in order to understand how to derive the confidence scores requested in the fourth column of Table 5.4, we refer to the standard technique in engineering of estimating a ‘triangular distribution’ [Ieee11a, p.297, OHJW04]. The more proficient modeller could specify a probability distribution that more closely matches their belief (e.g., normal,
beta, etc. [Alle06, p.206]), but the triangular distribution is considered to “a reasonable choice” [Alle06, p.203] where there is limited knowledge, despite its theoretical flaws [OBDE06, p.132].

Three parameters are estimated (Maximum, Mode, Minimum, — preferably in order) [BABD01, p.153] for a triangular distribution, and, by definition, values either side of the mode (assuming a unimodal distribution of belief) linearly decrease in likelihood, as is comparatively illustrated by Figure 5.22.

![Figure 5.22: Probability distributions (source: [Aust15]).](image)

Due to the existence of an ‘acceptability direction’ (i.e., higher or lower is better), the confidence score for a contribution link’s causal proposition (fourth column of Table 5.4) can be calculated as follows:

Given an XY graph with probabilities on the Y axis, possible extents of contribution on the X axis, three data points (Minimum, Mode, Maximum), and a line of linear interpolation between those points, then the confidence score can be calculated as the area under the interpolated line between the Mode and the Minimum or Maximum, depending on the variable’s direction of acceptability. This area can be calculated geometrically, given that the area of a triangle is equal to \((0.5 \times \text{triangleBase} \times \text{triangleHeight})\).

A number derived from frequentist probability is often used to represent confidence scores, i.e., the answer to such a question: if the requirement were implemented a large number of times, what percentage of those times would (at least) the stated contribution be contributed? The answer to such a question depends on the experiences of the stakeholders in implementing similar requirements in similar projects in a similar environment. However, similarity in this sense is difficult to achieve, since there are many socio-technical variables that can affect the benefits realised by a software project or a particular feature. Due to the fact that a software requirement can never be implemented in the same context a large number of times (since knowledge gained about the requirement cannot be removed from the minds of stakeholders), the frequentist interpretation is less applicable in this context. Hence, the descriptions of confidence could be better thought of as Bayesian probabilities, describing degrees of belief, rather than frequency or proportion [OBDE06].

A less rigorous, but more usable and accessible approach for eliciting confidence values under great uncertainty (where numerical confidence scores themselves would be almost wholly uncertain), is the use of textual representations of probability [OBDE06, p.197]. Hence, Table 5.5 provides an enumeration of qualitative confidence levels (based on the dimensions considered important by the Impact Estimation approach discussed in Section 4.8.2). Its use should be limited for stakeholders unwilling to provide numerical scores, and the corresponding numerical values should not be treated as though they had been directly elicited — they serve only to provide an indication of the effects of low confidence when paired with the Expected Value (EV) calculation.
Table 5.5: Confidence Level Enumerations to assist the modeller in choosing a confidence score under high uncertainty about the confidence score.

<table>
<thead>
<tr>
<th>Confidence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>Poor credibility, no supporting evidence or calculations, high doubt about capability</td>
</tr>
<tr>
<td>0.5</td>
<td>Average credibility, no evidence but reliable calculations, some doubt about capability</td>
</tr>
<tr>
<td>0.75</td>
<td>Great credibility, reliable secondary sources of evidence, small doubt about capability</td>
</tr>
<tr>
<td>1</td>
<td>Perfect credibility, multiple primary sources of evidence, no doubt about capability</td>
</tr>
</tbody>
</table>

(In Table 5.5, p=1 is considered as shorthand for p=0.99. Probabilities of 1 and 0 are often used to represent that an event certainly will or will not occur [OBDE06, p.1]. However, knowledge ‘about the world’ (as opposed to knowledge of ideas) can never be certain [HuHe95], or “There is no such thing as absolute certainty, but there is assurance sufficient for the purposes of human life” [MiGr08].)

An additional layer of confidence scores could be associated to the user’s predictions to represent their credibility. For example, someone who has implemented similar systems should be able to provide more accurate predictions than someone who has not. The accuracy of a person’s previous predictions should also be considered in order to improve the reliability of the predictions. (In the field of uncertainty elicitation, this is known as the calibration of subjective probabilities [OBDE06, p.61].)

An obstacle in eliciting and calibrating estimates from stakeholders, is that providing conservative estimates of benefit is often considered a safer strategy than providing ‘best guess’ estimates. This is will be especially frequent where an organisation uses estimates as commitments or for assurances. Essentially, this is a ‘What’s in it for me?’ problem, where there is little motivation to estimate high (realistic, but not totally confident) benefits, instead of low (but easily-achieved) benefits for a software feature. An organisation must be mindful of this, since the estimation process can easily become a useless ‘box-ticking’ exercise riddled with politics and self-preservation. (See also “Goodhart’s law”).

A low confidence estimate can be transformed into a high confidence estimate with increases in estimation effort [Jørg04]. One person’s opinion on a topic is quickly elicited, but can be made more robust by a survey of many people’s opinions (which should, ideally be informed by evidence and experience, and not pure speculation). Hence, there is the question about ‘what is confident enough?’. Scientific peer review often requires a confidence score of at least 0.95, which dictates the breadth of the confidence interval, and hence the degree of uncertainty. However, such questions are only adequately answerable by examining the consequences of an estimate being incorrect, rather than by following traditions and heuristics. In the words of Richardson & Pugh (on systems dynamics modelling), “To decide how much effort to put into estimating a given parameter value, one ought to know how sensitive the behaviour of the model is to the value of that parameter” [RiPu81, p.230]. Applied to the context of modelling benefits, the ‘sensitivity’ of the model to the uncertainty of a
contribution estimate, is determined by whether the ‘Target’ of an expected benefit is attained by the accumulated contributions of the software features (determined by the Expected Value calculation).

Finally, if an estimate has high variation due to factors other than lack of knowledge (i.e., due to aleatoric uncertainty), then rather than providing an estimate with a wide interval (or a point estimate with low confidence), an estimate could be provided for each variation-causing-scenario or ‘usage profile’. Each of these ‘usage profiles’ could then be associated with a weight number (such that the various usage profiles’ weights sum to 1), in order to describe the frequency of that usage profile relative to the others. For example, seasonal variation in demand for a process might cause such variation. (See this researcher’s paper attached to the appendix [Elli13] for a more thorough example).

### Stage 3/3: Multiple Point Contribution Estimation

As aforementioned, the single point contribution estimates made in Table 5.4 provide only a single XY data-point, and so if the ‘if’ clause (X) is not achieved (i.e., if the target level of satisfaction of the contributor is not met), then the ‘then’ clause (Y) is inapplicable and hence obsolete. Furthermore, the following are not represented by single point estimates:

- diminishing returns (non-linearity) in, for example, continuing to reduce a variable;
- effects of pessimistic, optimistic, and realistic extents of contribution;
- effects of incomplete (partial) benefit realisation or non-functional requirement satisfaction;
- effects of adding or removing a software feature on a benefit with multiple contribution links.

A ‘multiple point contribution estimate’ is essentially a set of ‘single point contribution estimates’, entailing a set of causal propositions covering different values of X for the ‘if X=x, then Y=y’ rules.

---

**Verification against the Required Capabilities for the Approach (Chapter 4)**

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

- ✔️ **[B]** Contribution-Extent-Score: “Be able to represent knowledge about the degree to which a means contributes to an end with either partial, full, or exceeding sufficiency and/or necessity.”
- ✔️ **[B2]** Contribution-Refutable: “Be able to represent contribution knowledge falsifiably and refutably, e.g., in terms of estimated impact on a numeric scale rooted in the application domain.”
- ✔️ **[B5]** Contribution-Uncertainty-By-Unknowns: “Be able to represent lack of knowledge (epistemic uncertainty) in the predicted contribution, i.e., the stakeholder’s degree of belief that implementing the requirement to some degree (the means) will lead to some degree of benefit (end).”
- ✔️ **[B6]** Contribution-Uncertainty-By-Variation: “Represent irreducible variation (aleatoric uncertainty) in the predicted contribution, i.e., both the extent and the reasons for any variation in a stakeholder’s estimate of the benefit caused by implementing the requirement.”

---

5.5 Estimate Predicted Contribution
The ‘if-then; elseif-then; elseif-then’ set of XY data points shown in Figure 5.24 is represented by defining a table function (as termed by Sterman [Ster00]), i.e., pairs of X and Y values, together with a chosen interpolation method (see Figure 5.25), where linear is chosen in Figure 5.24. This is a simple extension of the table used to define ‘single point contribution estimates’ in Table 5.4, where rather than each contribution link having one row, multiple rows can exist to describe different ‘if’ (X) values for the ‘if-then’ rules. (Table functions are used as opposed to analytic (i.e., algebraic) functions since analytic functions for non-linear curves are difficult to design, experiment with, and most importantly, communicate to stakeholders [Ster00].) A polynomial fit of the table function (or parametric function) could be computed for re-use, however, for simpler relationships (most likely linear), an analytic function could be described first-hand (e.g., $y=2x$), instead of a table function.

The table function should span the worst-case to the best-case range for each axis. If the value of X lies outside of the table function, i.e., X’s value is extreme, then ideally the table function should be updated to cover the extreme value, since the worst or best case points are no longer representative. Otherwise, the slope of the last two points could support extrapolation. Alternatively, the lowest and highest values of the table function could be mapped to all values lower and higher (respectively) than the table function’s range, in order to show finite potential for realising benefit after a certain numerical point.

The previously described confidence in each of the if-then rules (i.e., XY data points) can be visualised in figures such as Figure 5.25 in a number of ways. The simplest method is simply to increase the ‘thickness’ of the points and/or line as the confidence decreases, such that the graph’s interpretation is less precise. Alternatively, confidence intervals can be plotted, or a ‘funnel plot’ could be drawn.
Not all multiple point contribution estimates are between two expected benefits. Non-functional software requirements (such as [B] in Figure 5.13) are often specified with numeric targets, e.g., ‘...delete within 60 seconds...’; and it would be useful to describe the effects of changing that non-functional variable (‘60’ seconds to ‘120’, for example).

In simple cases where a non-functional requirement is explicitly related to an expected benefit with a contribution link, the method is essentially identical to the above. However, there are cases where the non-functional requirement is modelled to be necessary for the functional requirement with a decomposition link, and then that functional requirement is explicitly related to the expected benefit(s), such as in Figure 5.13. In such cases, each contribution link between a functional requirement (that has at least one non-functional requirement decomposed from it) and an expected benefit can be given extra attributes to describe the effects of changing the functional requirement’s decomposed non-functional requirements’ variables. Specifically, a coefficient can be estimated for each of the non-functional requirements to describe the effects of changing the non-functional variable on the functional requirement’s contribution. E.g., a positive coefficient of ‘1’ would mean that doubling the non-functional variable would also double the functional requirement’s contribution to the expected benefit. (Whether the contribution would be halved or doubled depends on the ‘acceptability direction’ of the non-functional variable, i.e., whether a smaller number is considered more, or less desirable.) Non-linearity in that coefficient can be described using a table function, as previously demonstrated, such that the ‘if’ variable represents the non-functional variable, and the ‘then’ variable represents the coefficient. This method ensures that if the contribution of a functional requirement changes, then the non-functional requirements’ contributions do not necessarily need to be changed, due to the relative definition of the effect on the contribution rather than an absolute definition.

Finally, a special type of multiple point contribution estimate is a ‘utility function’. Utility functions can be defined to answer the so-far unanswerable question of: “what is the benefit in achieving a root goal to various degrees of satisfaction?” (where the root goal is an expected benefit that does not contribute to other expected benefits, such as [F] in Figure 5.18). Essentially, a utility function should map various levels of a root goal’s satisfaction (e.g., in [F]’s case: various amounts of adverts produced per month) to various degrees of utility, defined on a 0-1 scale (where 0 is no utility and 1 is most utility). The root goals can then be weighted (i.e., where all of the root goals are given a number that sums up in total to 1), and then a total utility score can be calculated following the method described in Section 4.5.6. This ‘total utility score’ then describes the ‘goodness’ of a software system given any configuration of requirements, where changes to the requirement propagate up through the chains of instrumental value toward the root goal’s utility function. Utility scoring and weighting also enables ‘sensitivity analysis’, where stakeholders can understand the implications of small changes on the system.

<table>
<thead>
<tr>
<th>Verification against the Required Capabilities for the Approach (Chapter 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:</td>
</tr>
<tr>
<td>✔ [B3] Contribution-Dose-Response: “Be able to represent knowledge about the effect (the response) of modifying the means (e.g., the requirement’s target level of satisfaction)—the dose)—on an end.”</td>
</tr>
<tr>
<td>✔ [B4] Contribution-Non-Linear: “Be able to represent non-linear contribution in dose-response relationships, e.g., to show the scope for benefit maximisation and/or diminishing returns.”</td>
</tr>
<tr>
<td>✔ [E] Utility-Satisfaction: “Be able to represent the utility and disutility of satisfying or not satisfying an end, normalised (weighted) for comparison between the whole set of ends.”</td>
</tr>
</tbody>
</table>
5.6 Elicit Stakeholder Agreement

The degree to which a software requirement is believed to be instrumentally valuable, has thus far been estimated in terms of its contribution to expected benefits, which themselves are instrumentally valuable for other expected benefits. Despite that the estimates have been made refutably and in terms of objectively assessable domain phenomena, the estimates of the extent of contribution have been elicited based on subjective and personal belief (indeed, any model created by humans cannot be entirely free of subjectivity [Bued00]). Hence, in order for the software project’s stakeholders to truly be confident about its capability to be beneficial, different sources should be used, in much the same way that in the case study research method, conclusions should be ‘triangulated’ [RuHö08] from different data collection methods, and ultimately, different cases.

Groups of people tend to be far better at cognition (e.g., providing market valuations) and coordination (e.g., optimizing the utilization of a popular restaurant) than individual experts, on the condition that four criteria are adhered to [Suro05]:

- **Diversity of opinion**: individuals should be able to contribute information (even if is different interpretations of the same data) to the group, that was previously private to them;
- **Independence**: individuals’ opinions should not be determined by those around them;
- **Decentralization**: people are able to draw on local, context-specific, specialised knowledge;
- **Aggregation**: a mechanism exists for making a collective decision from private judgements;

Hence, a number of different people could provide estimates for one contribution link, and one ‘collective’ estimate can be generated. Research on mechanisms for aggregating estimates about an uncertain variable from numerous people, has shown that a simple average (mean) of their estimates tends to be equally as effective as more complex techniques [OBDE06]. However, given that stakeholder time is often scarce, it is important to prioritise the estimates that:

- have low confidence or low credibility, or;
- involve speculative predictions (e.g., estimates on the ‘increase in sales due to inclusion of feature X’ for market-based software), as opposed to calculated predictions (e.g., estimates on ‘the reduction in the time taken for process X, where constants Z, Y..., etc. are assumed to hold true) — i.e., for ‘anticipated benefits’ as opposed to ‘expected benefits’ (Chapter 2.1.3).

Finally, involving other people may well lead to more significant adjustments to the model, other than increasing the robustness of estimates, e.g., more optimal software features, or more expected benefits to realise. This activity is firmly in the spirit of RE, given that communicating the model to stakeholders to elicit their agreement is fundamentally a means of realising value from the model; Explicit models in RE have no value until they are rebuilt in the minds of stakeholders (i.e., mental models), so that they can assist understanding or highlight and communicate misunderstanding.

Verification against the Required Capabilities for the Approach (Chapter 4)
The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

[D1] Assumption-Agreement: “Guide the elicitation of stakeholders’ agreement (i.e., variation in belief) on the correctness of any assumptions made, including means-end contribution scores.”

5.7 Answer Questions

The information available from the model populated so far, is only as useful as the decisions that are influenced by it. Numerous use cases for this approach were identified in Chapter 2, but they all ‘boil down’ to requiring an answer to either:
a. What value would be provided to stakeholders if a requirement is implemented (or changed)?

b. What value would not be provided to stakeholders if a requirement is not implemented?

These two questions are not essentially the same, since they concern two alternative states of the world, whose agents would behave differently depending on whether the software requirement was implemented or not. The first question can be answered simply by examining all nodes connected to the software requirement, which can be queried and summarised automatically through the use of a graph traversal algorithm, such as ‘Depth First Search’ (DFS) (as implemented in the tool). Hence, given all of the requirements (or a user-defined selection of them), the satisfaction of the expected benefits (evaluated by their targets and their inward contributions) can be calculated automatically, thereby allowing a ‘red amber green’ style visualisation of each of the benefits’ status.

The second question concerns the behaviour of the modelled agents when the software agent does not exist. The ‘As-Is’ states are not guaranteed to remain the same in the future, and so there are numerous ‘As-Could-Be’ states. For example, in the context of the ‘Remove Expired Advert’ software feature frequently referred to throughout this chapter, the stakeholders might consider numerous other alternative ways of treating the ‘Removal Takes Too Long’ problem, e.g., hiring cheaper employees to remove the adverts, extending the length of advertised offers to reduce the number of adverts to remove, abandoning the use of adverts all together, and so on. It is an understanding of these alternatives that underpins an understanding of the value of a software requirement, i.e., relative to the alternatives. Indeed, the Betamax video format would likely still be valuable to millions of people around the world, if it were not for the existence of more valuable alternatives.

These alternatives to the software requirement (often non-software in nature) can be modelled as tasks owned by the agent who would perform them (just as in Section 5.3). Each alternative can be linked to the leaf level benefits using contribution links (just as in Section 5.4), since the default logical constraint on a contribution link is ‘OR’. Finally, answering question (b) now only requires an indication of the probability that each of the alternatives would be chosen (which should be determined by the strength of their contributions, both positive and negative).

Given that the dose-response cause-effect relationship of a contribution link has been defined (as guided in 5.5.1.2), then it is possible to understand the effects of changes in low-level elements (e.g., a software capability) on higher level elements (i.e., expected benefits) in a chain of instrumental value derivation. Only straightforward arithmetic on the as-is values of Expected Benefits needs to occur, such that the effects of the contribution links going to them are additively accumulated. Hence, stakeholders can use the model to answer questions such as ‘what happens if this Expected Benefit’s Target is only half realised?’, or more broadly, questions (a) and (b). See Appendix A for screenshots of the tool performing ‘What if?’ analysis and visualisation overlaid on the goal graph.

<table>
<thead>
<tr>
<th>Verification against the Required Capabilities for the Approach (Chapter 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ [F] Tool-Support: “Support the implementation &amp; ease the complexity of the approach with a software tool.”</td>
</tr>
<tr>
<td>✔ [F1] Tool-What-If-Answering: “Support the answering of ‘To what extent are directly and indirectly related benefits affected if the requirement is adjusted?’ questions using a software tool.” – see Appendix A for screenshots/walkthrough of the tool.</td>
</tr>
</tbody>
</table>
5.8 Post-mortem & Re-use

The closing of a project is considered to be the most important phase to identify and capture knowledge for transfer to other projects [Dist02]. The terminology used in related approaches for project post-mortem indicates the intention of this phase: Post project review; After action review; Reuse planning; Cooperative project evaluation; Reflection; Corporate feedback cycle; Post implementation evaluation. These approaches are concerned with qualitative ‘lessons learned’ evaluation, with the exception of Basili’s more quantitative ‘Experience factory’ [Basi93c]. Importantly, none refer to the retrospective validation of a project’s requirements as hypotheses for realising some degree of benefit.

There are two classes of information that we are primarily interested in re-using following post-mortem:

- Means-end knowledge about an end being achievable by a means in a given context;
- Means-end contribution strength (the degree to which the means achieved the end), as evidence to inform future estimates of contribution strength for similar future means’.

Even without a project post-mortem to understand the actual effects of the requirements on the expected benefits, the prescriptive and predictive aspects of a project’s model might be useful for future projects. For example, while the software engineers in this running-example project were able to propose the software feature ‘delete media automatically (on a schedule)’, there is no guarantee that future software engineers in different projects would propose it. Figure 5.26 provides a further example of potential re-use from a previous project (see the dotted arrow in the bottom right corner). (Note that the same software feature is proposed, but for different reasons. Without having explored these project-specific reasons, a developer might not understand why the feature should be developed, and so might not have proposed it in the first place, or might choose not to implement it.)

![Figure 5.26: Re-use of models from previous projects (here, a new requirement is suggested).](image)
In terms of recommendation systems (machine learning), such a recommendation system is classed as a ‘document’ recommendation system [AJDE07], drawing heavily on Information Retrieval techniques.

This researcher has created an approach for automatically assessing the similarity between problem requirements and solution requirements, using state of the art information retrieval techniques. For brevity, a full technical description of this technique can be found in the paper [ELDK14] (included in this thesis’ Appendix). This technique is straightforwardly applied in the approach’s tool for recommending and re-using model fragments, by considering expected benefits and software requirements both as types of requirement whose similarity should be assessed.

Owing to the impact of project context on benefits, a secondary similarity score is required for efficient re-use of data (i.e., to improve ‘precision’ rather than ‘recall’). For example, even if the software is unchanged, a digital signage system for a hospital would have different expected benefits (e.g., keeping patients informed about appointment times) than it would in the running example’s context. An assessment of the similarity of each project’s ‘Object’ fields (defined in the structured expected benefits template), as well as the associated actor positions/roles, provides a way of matching model elements from the most relevant previous projects. A TF-IDF (term frequency-inverse document frequency) score calculation of these terms (where one project’s set of terms acts as one ‘document’) provides the mechanism for this comparison between project contexts.

Risk related information can also be re-used. E.g., given the knowledge codified in the models produced in this walkthrough, the tool would be able to prompt the requirements engineer to evaluate the risk that the obstacle ‘[Graphic Designer] Not Sufficiently Motivated’ (in Figure 5.20) applies, whenever they create (or select) an element in the tool that is similar to an element linked to that obstacle (e.g., one containing the words ‘Production Output’ as in Figure 5.20). Parent-child, as well as other relationships (sibling, cousin, grandparent, and so on) can be traversed to provide such recommendations to the modeller about adding to, or modifying the model. Screenshots of the re-use functionality in the framework’s supporting tool (as well as others supporting the framework), such as Figure 5.27 (see bottom-right corner), are included in the Appendix.

![Figure 5.27: Screenshot of the software tool showing the information retrieval capabilities in the lower right hand corner (using a highly simplified example to aide comprehension of the image).](image-url)

**Verification against the Required Capabilities for the Approach (Chapter 4)**

The following requirements for the approach (defined in Chapter 4) have been satisfied by this section:

- ✔️ [F2] Tool-Reuse: “Support the elicitation of a requirement’s benefits and disbenefits (outlined in criteria A & B) via automated retrieval & re-use of relationships within previous project models.”

192

5.8 Post-mortem & Re-use
5.9 Summary

This chapter has proposed a framework for modelling the instrumental value of a software requirement that meets the criteria discussed in Chapter 4.

While the models may seem daunting even for relatively straightforward software features, this highlights the complexity of value often overlooked by software projects. If instrumental chains are not explored, then there is the risk that the software capability or shallowly-defined rationale or benefit will not derive as much value as the stakeholders expect from its implicit sources of value. If there is ignorance on the numerical aspects of the instrumental value models (even in mental models constructed in the same fashion as this framework’s explicit models), then there is the risk that stakeholder expectations about the degree of value to be realised by the software capabilities are incorrect. In other words, an unpopulated (i.e., incomplete, abstract, or overly qualitative) model is not necessarily uninformative, in that it makes explicit the lack of knowledge about the value expected and to be realised, and hence highlights risks and areas of ignorance that the software project team should research. Note that, in order to judge the progress in constructing the models advocated in this approach, simple ‘coverage’ and ‘completeness’ percentage metrics can be calculated (e.g., ‘how many leaf software requirements are traced to expected benefits?’, or ‘on average, how many of the expected benefit hardness fields are populated?’).

To gain at least some advantage from this framework, it is not necessary for a project to explicitly model the instrumental value of its software requirements. The sentiments of Howard are shared in paraphrasing his conclusion on decision analysis: This framework is ‘not just about learning the subject or using the tools’, but rather “more like installing a new operating system in your brain” [Howa07, p.40] through its principles and philosophies on software. In other words, the key point from this framework is that a requirements engineer should think precisely about the various causal levels of an uncertain quantity to be influenced in an application domain that would actually entail the realisation of stakeholder value, whenever a system capability is planned. When it comes to explicit modelling, a useful parallel can be drawn to evaluations of PERT, which indicate that its “major benefit” is derived from its linear cause-and-effect style “network analysis” of activities, rather than through its estimation of activity completion time [Awat08, Sapo72]. I.e., its value is in “structured thinking” and subsequent discussion, rather than in the numbers resulting from the “almost always” inaccurate estimates [Galw04]; Similarly, the major benefit of the numeric scales in this approach is likely in that they help to unambiguously define the structure of the goal graph.

Finally, quotes such as “I have always found that plans are useless, but planning is indispensable” have some currency, but with the use of automated information retrieval and recommendation techniques, yesterday’s plans can become highly useful for tomorrow’s planning.
6. Case Studies

“Theory and reality are only theoretically related.”
Robert Grossblatt
Chapter 2’s review of the literature established that numerous software requirements are developed only to be left unused. However, empirical data supporting that claim is sparse, and even more so, data exploring its causes and the state of practice in preventing it. This is essential for motivating and evaluating the framework established in the previous Chapter 5, and so this chapter aims to do what too little research in RE does [MaMa13], by exploring RE practice rather than theory.
Results of questionnaires, interviews, ethnographic research, case examples, expert opinion, and document analysis, are presented in order to contribute to answering the following research questions (defined in Chapter 3), as is shown by Table 6.1.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Content in this Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ1.2: What is the motivation for analysing the value of software requirements [elicited via primary research in industry]?</td>
<td>Questionnaire, Interviews</td>
</tr>
<tr>
<td>RQ1.3.2: How is the value of software requirements currently modelled [elicited via primary research in industry]?</td>
<td>Questionnaire, Document Analysis, Interviews</td>
</tr>
<tr>
<td>RQ1.3.3 How effective is that value modelling?</td>
<td>Document Analysis, Questionnaire, Interviews</td>
</tr>
<tr>
<td>RQ2.2: What are the causes of software requirement value realisation failure?</td>
<td>Questionnaire, Interviews</td>
</tr>
<tr>
<td>RQ2.4: What features would be useful in an approach to model the instrumental value of software requirements [elicited via primary research in industry]?</td>
<td>Interviews</td>
</tr>
<tr>
<td>RQ4.3.2: Verify and validate the framework [via primary research in industry]?</td>
<td>Expert Opinion, Case Examples</td>
</tr>
</tbody>
</table>

6.1 Questionnaire

This section introduces and provides the design rationale for the survey distributed to this thesis’ industrial partner organisations (Sections 6.1.1 through to 6.1.5), and then discusses and evaluates its results (Section 6.1.6).

6.1.1 Rationale for the Online Survey Technique

In order to elicit perceptions on the aforementioned research questions from industry practitioners, an online questionnaire was constructed. This technique has almost entirely superseded traditional survey techniques (e.g., postal, in-person) for its ability to reduce errors (of coverage, sampling, measurement, and non-response) [Umba04], increase respondent anonymity and flexibility, and reduce cost and time consumption (of distribution, collection, codification, and analysis) [DiTB98]. The common disadvantages of online surveys [Bowk01] have been considered, and are not considered significant for this thesis’s context:

i. **Technical challenges of creating and administering the survey.**
   Survey tools such as Bristol Online Surveys [Bris13]54 have made the technical effort required low, compared to the researcher having to code or install their own web form and database;

ii. **Non-response due to indirectness of communication method.**
   In both organisations, the respondent’s line manager (or higher) will invite the respondent to participate, in order to improve directness and authority of communication;

iii. **Ethical and privacy concerns about the respondent, and ‘spam’ invites.**
   The survey will be anonymous unless the participant chooses to provide contact information. The respondent’s line manager will approve the survey (and related communication) for its relevance and appropriateness to the organisation. Financial incentives will not be offered.

Finally, Table 6.2 presents an evaluation of common concerns for the four aforementioned survey error types related to the use of online questionnaires, as is suggested best-practice [Umba04].

---

54 ‘Bristol Online Surveys’ is the standard online questionnaire service at this researcher’s institution, having agreements in place for data security and reliability to allay the organisations’ data security concerns.
Table 6.2: Relevance of Survey Error Types to the Planned Questionnaire

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Concern for Online Survey?</th>
<th>Applicable to the two industry Case Studies?</th>
<th>Problem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage Error</td>
<td>Respondent does not have access to computer or internet.</td>
<td>All potential respondents have access to (and skill for) the online survey due to the technical nature of their jobs.</td>
<td>No</td>
</tr>
<tr>
<td>Sampling Error</td>
<td>A random sample requires that the target population is capable of completing the survey.</td>
<td>The organisations are large, so a census would not be feasible with the research project’s resources. Online surveys allow the sample size to be affordably large.</td>
<td>No</td>
</tr>
<tr>
<td>Measurement Error</td>
<td>Clarification of meanings is not immediately available. Answers may be skewed since online participants may be more technologically savvy.</td>
<td>One-to-one sessions will be offered for those confused by questions. Since all respondents will be savvy enough to respond online, enthusiasm for research is a far bigger skew concern.</td>
<td>No</td>
</tr>
<tr>
<td>Non-Response Error</td>
<td>Certain social groups may be under represented among internet users.</td>
<td>The population is already biased (gender &amp; age) toward older males in engineering [Onwu13]. This is unlikely to be exaggerated by an online survey, as discussed in 'coverage error'.</td>
<td>No</td>
</tr>
</tbody>
</table>

6.1.2 Rationale for the Questionnaire’s Design

The extensive literature on best practice in online questionnaire design is summarised by Umbach’s checklist in Table 6.3. Adherence to this best practice is claimed to improve response rates, and the questionnaire’s ability to answer research questions. Hence, Table 6.3 also evaluates the survey design’s adherence to it. (The resulting survey questions and participation invitations, as the respondents saw them, are included in the appendix.)

Finally, despite that Table 6.3’s best practice list is an aggregation of many sources, it is not exhaustive, and so other best practice guidelines were found and applied when design uncertainty existed (e.g., grids versus individual questions [CoTL01]).
<table>
<thead>
<tr>
<th>Design Principle</th>
<th>Potential Error Type (Non-Response, Sampling, Coverage, Measurement)</th>
<th>Implementation Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Provide a password to limit access to only those people in the sample.</td>
<td>Sampling, Coverage</td>
<td>✔ A password would hinder usability and anonymity; The URL’s provided to the participants will not be publicly listed.</td>
</tr>
<tr>
<td>3. Make the first question interesting, easily answered, and visible on the first screen of the questionnaire.</td>
<td>Non-Response</td>
<td>✔ The first question is simple (number of software projects in the last 10 years), and forces the respondent to think about their projects — framing later questions. It is only interesting in that most people enjoy talking about themselves [Carn88].</td>
</tr>
<tr>
<td>4. Use a conventional format similar to paper questionnaires.</td>
<td>Measurement, Non-Response</td>
<td>✔ Best practice for designing surveys [CoTL01] is complied with.</td>
</tr>
<tr>
<td>5. Unless order effects are important, allow respondents to scroll from question to question.</td>
<td>Coverage, Measurement, Non-Response</td>
<td>✔ Implemented by default in software.</td>
</tr>
<tr>
<td>6. Avoid question structures that have known problems in traditional surveys (e.g., (open-ended, biased, leading) questions).</td>
<td>Non-Response</td>
<td>✔ Best practice for designing surveys [CoTL01] is complied with.</td>
</tr>
<tr>
<td>7. Use a welcome screen that is motivating and assures confidentiality.</td>
<td>Non-Response</td>
<td>✔ The first page describes the aim of the survey and confidentiality.</td>
</tr>
<tr>
<td>8. Avoid the use of drop-down boxes.</td>
<td>Measurement</td>
<td>✔ Radio buttons are used for response options that are mutually exclusive.</td>
</tr>
<tr>
<td>9. Limit line length to make it less likely that respondents will skip words.</td>
<td>Measurement, Non-Response</td>
<td>✔ Each question has been refactored to minimise verbosity and ambiguity.</td>
</tr>
<tr>
<td>10. Provide clear and specific instructions about how to navigate through the survey and how to answer questions.</td>
<td>Measurement, Non-Response</td>
<td>✔ Sequential question numbering is provided. Examples and references are provided to explain technical concepts.</td>
</tr>
<tr>
<td>11. Do not include a counter that measures website visits.</td>
<td>Coverage, Non-Response</td>
<td>✔ Implemented by default in software.</td>
</tr>
<tr>
<td>12. Test the survey on different computers to avoid differences in the visual appearance resulting from differences in computers.</td>
<td>Coverage, Measurement, Non-Response</td>
<td>✔ Implemented by default in software.</td>
</tr>
<tr>
<td>13. Include a progress timer to indicate to respondents approximately how much more time it will take to complete.</td>
<td>Coverage, Non-Response</td>
<td>✔ Estimated time to completion is provided on each page.</td>
</tr>
<tr>
<td>14. Do not require answers to each question before allowing respondents to continue with the survey.</td>
<td>Coverage, Non-Response</td>
<td>✗ This is not a concern since all of the necessary questions are on one page, for visibility of progress and future questions.</td>
</tr>
<tr>
<td>15. Divide long surveys into sections.</td>
<td>Coverage, Measurement, Non-Response</td>
<td>✔ The survey is grouped into sections of questions on the same theme.</td>
</tr>
</tbody>
</table>

### 6.1.3 Rationale for the Design of the Questions

The questions probed the participant’s previous involvement in software projects during the last 10 years. This limit was chosen for the following reasons:

1. Some employees have more than 40 years of professional experience. Asking them to recall that far back would take too much time (and so they would probably guess instead).
2. The 10 year interval (i.e., from 2003 to 2013) saw significant advances in the literature and in practice (e.g., in 2005, the Value Based Software Engineering agenda was outlined [Boeh05], and Agile’s popularity significantly grew [Scot07]). By 2003, eight more years of evidence had
been collected to support Standish’s original (1995) claim that half of implemented software features are never used in organisational IT systems after their implementation [John03].

Despite the consequences of aggregation, participants were surveyed on their experience on many projects, rather than a single project (e.g., their last completed project) for the following reasons:

1. Questions relating to the realisation of value from software require that the developed or procured software has been deployed, is in use, and that users are comfortable with it; The benefit of many IT systems is not realised until many years after their deployment [Thor99].
2. While the depth of the information would be improved if responses were specific to one project, at this first stage of research, breadth is preferred. Follow-up structured interviews will later be performed to capture detail.

According to Curwin and Slater [CuSl07, p.68], questions can be used for eliciting a respondent’s:

1. Awareness of an issue (e.g., are you aware of X?);
2. Feelings on an issue (e.g., should X be built?);
3. Perceived understanding of an aspect of an issue (e.g., will X will affect Y / how often Xi?);
4. Reasons for a view on an issue (e.g., why are you against X?);
5. Importance placed on the reasons for a view (e.g., [reason against X] is 7/10 important).

Each of these five concerns are covered by the questions most pertinent to the research questions. Importantly, before asking participants for information on a specific aspect of an issue (3) (e.g., Q4: ‘on average... how many functional requirements were redundant?’), their ability to answer it (and hence the robustness of the response) was elicited by asking for their awareness (1) of the pre-requisites, (e.g., Q3: ‘how many projects do you know of the usefulness of implemented requirements?’).

One especially noteworthy design decision is the use of textual quantity labels on an ordinal scale in the pre-set answers, as opposed to numbers on an interval or ratio scale. In an ideal world, respondents would be able to provide precise percentages, for example, for the average proportion of functional requirements that turn out to be redundant in software (as the CHAOS survey supposedly presumes). However, in a pilot run of an initial version of the questionnaire that requested answers as numerical percentages, most respondents found the implied precision (even in interval pre-set answers) to be off-putting. Indeed, the question implicitly asks the respondents to recall each project within the last 10 years, recall the requirements set and the proportion of features actually used by end-users, and then average each result to arrive at the answer\textsuperscript{55}. It would be farfetched to assume that busy industry practitioners would go through such a thorough thought exercise for a questionnaire that they are not obliged to complete, nor that benefits them in any way. Hence, the options of \{Almost none, Some, Around half, Most, Almost all\} were chosen as pre-set answers in recognition of the ‘fuzziness’ inherent in the information that would most likely be provided. Nevertheless, the labels are roughly evenly spaced from 0% to 100% assuming 25% linear increments, and crucially, the scale that these pre-set answers refer to is objective and cross-comparable (as opposed to a Likert scale of agreement). Finally, and pragmatically speaking, the first two of those labels are sufficient for motivating work on the problem (i.e., to support the claim that the problem exists in practice), and the third label (‘Around half’) is sufficient for eliciting agreement with the CHAOS report’s claim that half of required software features are unused.

\textsuperscript{55} To save time, the respondent might choose one ‘typical’ project (i.e., the modal project type) to provide an average, which would skew the results where that modal type is not representative of the set of projects.
The ordinal question responses included a ‘Don’t Know or N/A’ option so as to improve the fidelity of the response set. Studies show that excessive and inappropriate choice of the ‘middle point’ response (e.g., ‘around half of the projects’) is reduced when a ‘Don’t Know’ option is provided [OBDE06, p.86]. In other words, people often use ‘50-50’ to represent their lack of knowledge, even when ‘50-50’ actually implies a frequent occurrence.

6.1.4 Rationale for the Questionnaire’s Construction, Sampling, and Distribution

The survey questions were derived from the research questions in order to ensure their pertinence. The questionnaire had one primary, but not restrictive aim: to establish an understanding about the proportion of software requirements that lead to unused software features. It was made clear to the participants that the survey was for a PhD research project on modelling the strategic alignment of software requirements. (As established in the conceptual framework, ‘strategic alignment’ is essentially the business terminology for the relationship between different planned means and longer-term desired ends. It was used instead of ‘instrumental value’ since the participants were more likely to have a business than philosophy background.)

The questionnaire was reviewed regularly during its development, primarily at Case B (defence IT consulting company), and then subsequently at Case A (aeronautical engineering company) to avoid redundant feedback. Before the questionnaire was sent to the sample population, three successive pilot runs were conducted with experienced practitioners to gain an insight into how the questions would be answered. The questionnaire had company director level approval at Case B (due to the organisation’s interest in RE and good relationship with the researcher’s institution), and department head (Component Design Systems) approval at Case A.

While a random sample of a well-defined population would be ideal for ensuring representativeness of the survey results, due to resource constraints it was not possible to census the whole population of requirements engineers at the organisations — a precursor for maintaining random sampling’s constraint that “each unit in the population has an equal chance to be drawn” [BeFr03, p.1]. Indeed, for pure statistical inference (generalising from samples to populations using statistical tests) as opposed to logical or analytical generalisation, this ‘random sampling assumption’ is required, but Berk & Freedman conclude that this assumption has “substantive implications that are unrealistic” [BeFr03, p.4]. Therefore, case study (and even laboratory [Warn12, p.95]) researchers often acceptably work with, and apply statistical inference to, so-called ‘convenience samples’ [Farq12, p.71].

Consequently, a list of experienced requirements engineers for software systems was identified at both organisations with the assistance of the research project’s industry liaisons, in order to form an intended ‘representative sample’ [Busi14b] for the construction of a ‘typical case study’ (of 9 possible types) [Gerr06, p.90]. Formally, the non-probability sampling methods of ‘judgement sampling’ and ‘snowball sampling’ were employed [HWMS15, p.176]. Both samples were also guided by the principles of ‘quota sampling’ [HWMS15, p.176] in order to provide a variety of experience, and more crucially to avoid a homogenous pool of projects that the responses refer to. Hence, Case A’s sample spanned a variety of its business units, from Supply Chain Units (SCUs) to service departments such as its ‘Software Centre of Excellence’, while Case B’s sample spanned both its business analysis and software development business units.

A final list of 42 engineers at Case A and 23 consultants at Case B was identified for participation in the survey. The required sample size was determined to be statistically appropriate using the authoritative guidelines in [Mccr10], or more specifically according to the ‘Sampling Size of
Population Proportion equation [Raos04, Yauc13a, Yauc13b, chap.10]. An invitation email was distributed (included in the appendix), followed by a reminder 2 weeks later, followed by individual reminder emails. The participants were told in the invitation email that the survey was anonymous, that the results would not be used for internal (company) benchmarking or other commercial uses, and that they would not be published publicly without removal of sensitive information. The equation for sample size adequacy was calculated throughout the survey’s distribution in order to be able to increase the sample size if necessary, given that the sample size required to reach reliable conclusions by statistical tests depends on the distribution of responses to individual questions, as well as the required proportion of responses to reject a survey question’s null hypothesis [Raos04].

Case B’s questionnaire had a 70% response rate (16/23), while Case A’s had 26% after removal of an invalid response (11/42) (—one respondent reported experience of 0 software projects). Case A’s lower response rate (26%) is not out of the ordinary; According to Conford and Smithson, surveys sent out without prior communication with the participants have an average response rate of 20% [CoSm96]. The difference between the response rates can likely be explained by the fact that Case B’s had endorsement from a highly senior employee (and, who was copied into the invitation emails), and that, as previously reported by Ubhi in a previous research project at Case A, Case A’s employees are often overwhelmed by voluntary survey invitations [Ubhi08].

<table>
<thead>
<tr>
<th></th>
<th>Case A</th>
<th>Case B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry</td>
<td>Aeronautical Engineering</td>
<td>Defence Contractor</td>
</tr>
<tr>
<td>Military/Civilian projects</td>
<td>Mostly civilian</td>
<td>Mostly military</td>
</tr>
<tr>
<td>Type of software produced</td>
<td>Component Design Automation (Geometry creation and analysis)</td>
<td>Information Management; Data Analysis</td>
</tr>
<tr>
<td>Location</td>
<td>East Midlands, UK</td>
<td>West Midlands, UK</td>
</tr>
<tr>
<td>Number of employees</td>
<td>&gt; 50,000</td>
<td>&gt; 25,000</td>
</tr>
<tr>
<td>(Yahoo Finance 2015)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Questionnaire response rate</td>
<td>11/42 (26%)</td>
<td>16/23 (70%)</td>
</tr>
<tr>
<td>Questionnaire respondents’ years of software engineering experience</td>
<td>Median: ≤15 years</td>
<td>Median: ≤15 years</td>
</tr>
<tr>
<td></td>
<td>27% reported ≤5 years</td>
<td>6% reported ≤5 years</td>
</tr>
</tbody>
</table>

6.1.5 Rationale for the Analysis of the Questionnaire’s Responses

The responses to the questionnaire were analysed and are reported with compliance with Kitchenham and Charters’ guidelines for empirical studies [KiCh07] (a compliance checklist table is not presented since there are over 50 guidelines). Hartman’s decision tree for choosing analysis methods [Hart00] was followed, and so the reader is referred there if they are curious as to why a particular method was used. A shorter version is shown in Table 6.5 for illustration, (it is expanded by, for example, suggesting the non-parametric Wilcoxon rank sum for testing ordinal (and not necessarily normally distributed) data [OtLo15, p.324], instead of suggesting the parametric t-Test, given that it is incorrect to compare the sample means of ordinal data).

Definitions of statistical terms used in this section and henceforth throughout this chapter are as follows:

The p-value is the probability that we will obtain a difference in sample estimates as large as the difference we have, if the null hypothesis were true [e.g., if there is no significant difference] [PaCT95, p.198]. p-values do not measure support for alternative hypothesis, but low p-values support rejection
of the null, which gives reasonable evidence to support the alternative [AIWZ10, p.461, Fros14]. A correlation coefficient is “a measure of the strength of dependence between two variables” [Stat00a]. A 95% confidence interval is “the interval that you are 95% certain contains the true\(^57\) population value as it might be estimated from a much larger study” [Stat00b]. A standard deviation (reported as SD [Kahn11]) expresses the degree to which a group’s members differ from the group’s mean value. Finally, significance is distinct from statistical significance, in that “a difference to be a difference must make a difference” [PeSc13, p.203]. I.e., a statistically significant finding is of no real consequence if the effect size is miniscule, or in other words, if the finding is insignificant to the world.

Table 6.5: Short version of Hartman’s decision tree for choosing statistical analysis methods [Mare07]

<table>
<thead>
<tr>
<th>Statistical Analyses</th>
<th>Independent Variables</th>
<th>Dependent Variables</th>
<th>Control Variables</th>
<th>Question Answered by the Statistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi square (X(^2))</td>
<td>1 categorical</td>
<td>1 categorical</td>
<td>0</td>
<td>Do differences exist between groups?</td>
</tr>
<tr>
<td>t-Test</td>
<td>1 dichotomous</td>
<td>1 continuous</td>
<td>0</td>
<td>Do differences exist between 2 groups on one DV?</td>
</tr>
<tr>
<td>ANOVA</td>
<td>1 + categorical</td>
<td>1 continuous</td>
<td>0</td>
<td>Do differences exist between 2 or more groups on one DV?</td>
</tr>
<tr>
<td>ANCOVA</td>
<td>1 + categorical</td>
<td>1 continuous</td>
<td>1 +</td>
<td>Do differences exist between 2 or more groups after controlling for CVs on one DV?</td>
</tr>
<tr>
<td>MANOVA</td>
<td>1 + categorical</td>
<td>2 + continuous</td>
<td>0</td>
<td>Do differences exist between 2 or more groups on multiple DVs?</td>
</tr>
<tr>
<td>MANCOVA</td>
<td>1 + categorical</td>
<td>2 + continuous</td>
<td>1 +</td>
<td>Do differences exist between 2 or more groups after controlling for CVs on multiple DVs?</td>
</tr>
<tr>
<td>Correlation</td>
<td>1 dichotomous or continuous</td>
<td>1 continuous</td>
<td>0</td>
<td>How strongly and in what direction (i.e., +, -) are the IV and DV related?</td>
</tr>
</tbody>
</table>

It is important to stress that the results of such inferential statistical analyses are by themselves insufficient for generalising to a theory or population, due to the violation of many of the analyses’ assumption of random sampling (as discussed in Section 6.1.4). (The applicability of inferential statistics on nonprobabilty samples is a common concern [BrDr13, PeSc13, p.198, ThDr03], e.g., case-control studies have no randomization but p-values are almost always reported [OkDJ14]. Much of this is rooted in the belief that p-values are invested with “magical powers” of conclusion, instead of being merely an “index of surprise” [PeSc13, p.198].) Case study generalisations should be based upon ‘common-sense inferences’ (i.e., analytical and logical generalisation) dependent on the plausibility of the argument, as opposed to pure statistical generalisation [Must08, p.15]. Nevertheless, Gerring stresses that “the introduction of statistical analysis does not — should not — disqualify a study as a ‘case study’”, and that doing so is popular for enabling a less textual and more ‘synthetic’ analysis [Gerr06, p.33]. Hence, despite that p-values are reported in this chapter in the standard parlance of rejecting or supporting the null hypothesis, they are intended to merely support conclusions in compliance with Mackay’s advice to “use statistics as a drunken man uses lamp-posts — for support rather than illumination” [Mack77, p.91].

Finally, the R statistics software [Rcor14] was used for hypothesis testing and data exploration, in accordance with Dalgaard’s guidelines [Dalg08]. Amongst R packages such as corplot, fBasics, and coin (since R’s wilcox.test() cannot compute exact p-value with ties), the rattle exploratory statistics R library was used to expose relationships and trends within the responses. If it were not for exploratory statistics, researchers would only have intuition and often biased expectations to use when selecting variables to examine, and since many pairs of variables exist, relationships may go unnoticed.

\(^57\) I.e., that which hypothetical repetition of the survey (testing for the impact of chance) would find.
Figure 6.1 shows a correlation plot between the responses-of-interest for the questionnaire results in order to illustrate one type of exploratory data analysis used. Correlations that might explain a survey response, or are pertinent to one of the research questions, are discussed in-line with the response discussions, and so the reader need not inspect the text of Figure 6.1. Note that most statistically valid findings from exploratory statistical analysis will be spurious, regardless of the strength of \( p \)-values, correlation coefficients, confidence intervals, and so on. [Nuzz14]. As such, the researcher was careful to consider hypotheses for their plausibility, and to avoid the ‘Texan Sharpshooter’ fallacy: ‘A cowboy shoots hundreds of gunshots into the side of a barn, paints a bulls-eye over the biggest cluster of bullet holes, and then claims to be a sharpshooter’ [Gawa98].

![Figure 6.1: Correlation plot for the questionnaire questions of primary interest for the combined response set (both organisations’ responses). The X-axis shows Kendall’s rank correlation tau. The white crosses represent insignificant correlations (significance test \( p \)-values greater than 0.1). The question numbers refer to Case B’s questionnaire for traceability (both included in the Appendix). [The \( p \)-values quoted for correlation tests are for the alternative hypothesis that the true tau is not 0.]]
Finally, it should be noted that comparisons of Case A & B’s responses to the questions are not to conclude that either case is in some way better or worse; Even if this was desired, it would not be possible given that the data set is comprised mostly of aggregated data, resulting from questions beginning with ‘on average over your projects’. E.g., if Case A achieves a higher benefits realisation rate than B, then it might be that B takes on riskier projects; B may in fact be ‘better’ than A for projects of a certain type. In summary, there are a myriad of ‘lurking variables’ that the survey does not, and could not practically exhaustively expose (see Simpson’s Paradox [AIWZ10, p.208]), that could undermine conclusions of either organisation’s superiority in some aspect of RE practice.

### 6.1.6 Questionnaire Responses

This section analyses the responses to the questionnaire’s 19 questions. The questionnaire was first distributed to Case B’s employees and then to Case A’s, both isolated and independent from each other. Both sets of results are shown for each question to reduce redundancy and repetition. In some questions the combined set of results is also analysed to provide a wider overview.

#### 6.1.6.1 How many software development projects have you contributed toward in the last 10 years? (Case A: Q3, Case B: Q1)

Case A’s respondents reported a mean experience of 6.3 software projects (SD=3.8, Median=5) in the last 10 years, while Case B’s respondents reported a mean of 14.7 projects (SD=8.3, Median=15). A total of 310 projects is represented by the combined responses (75 projects by Case A, and 235 by Case B), and while these 310 projects are unlikely to be entirely distinct nor independent from each other, the representative sample was selected to maximise diversity and to avoid a homogeneous project experience base. E.g., both sets of respondents were selected from a variety of the organisation’s business units and from a variety of subject matter specialisms, although Case B’s projects (consulting) are likely to be more diverse since the projects’ customers belong to other organisations.

![Figure 6.2: Number of software development projects contributed toward in the last 10 years — distribution of responses](image)

One might assume that it is better for the survey’s data quality for the participants to have experienced a higher number of projects (as in Case B). However, experience of fewer projects is also valuable, since these participants may have had less social ‘conditioning’, (e.g., assuming that widespread bad practice is normal and acceptable), and are more likely to have deeper exposure. The latter is crucial, since the participants’ knowledge of the projects’ various software development lifecycle stages (at least ‘development’ and ‘usage’) is required for describing the usefulness of implemented requirements.

The more-than-double difference between Case A & B’s mean respondent experience (6.3 projects vs. 14.7 projects — treating the response intervals as lower bound estimates) either indicates dissimilarity between their projects’ durations, or suggests two types of respondent experience:
• ‘deep and narrow’: e.g., developers tasked with long-term work for a small number of projects;
• ‘wide and shallow’: e.g., subject matter experts tasked with short term work on many projects.

The above distinction and implication that Case B has more participants in the ‘wide and shallow’
group is plausible given Case B’s business context (consulting) and its associated nature of work,
often entailing ‘chunks’ of projects specialised to a particular role. Indeed, later results shown in
Figure 6.7 support the theory that Case A’s participants have deeper (‘through life’) knowledge about
their projects than Case B’s.

Takeaway Point #51: Consulting has higher project turnover than single-organisation work

Case A’s participants were involved in fewer projects than Case B’s in the same 10 year period (0.63 vs. 1.47 projects per year).
Hence Case B’s participants should know about twice as many ‘matured’ projects (to comment on their usefulness).

6.1.6.2 How many of these projects (from 6.1.6.1) do you know of that are now (or were) in use by the end-users? (Case A: Q4, Case B: Q3) ([Don’t Know], Almost none, Some, Around half, Most, Almost all)

The responses to this question and the next two determine the size of the participants’ project
experience base that is suitable for answering this thesis’ research questions, given that a requirement’s
usefulness (or wastefulness or inadequacy) cannot be known until it is implemented and used.

![Figure 6.3: Proportion of the respondents’ projects that are now (or were) in use — distribution of responses](image)

It is evident from Figure 6.3 that Case A’s respondents knew of a higher proportion of software
projects that progressed to ‘in-use’ than Case B’s respondents. Indeed, 81% of Case B’s respondents
reported ‘Most’ or ‘Almost all’ projects, compared to Case A’s 57%. However, the difference between
Case A & B’s medians is not statistically significant — both being ‘Most’ projects ‘in use’.

Despite that it ‘Most’ projects appears to have progressed to ‘in-use’, it cannot be ignored that some
respondents (Case B: 25%, Case A: 18%) knew of less than half of their projects’ software that did
so. Therefore, roughly a quarter of both cases’ respondents had below half of their projects that either:

i. were still in development at the time of the survey, as-yet not released to the users, or;
ii. never resulted in the software being used (i.e., cancelled or wholly redundant projects), or;
iii. entailed strictly ‘development only’ (i.e., not ‘usage’) involvement & knowledge.

If (i) were true, then one would expect the < ‘Around half’ respondent group to have fewer years of
software development experience than the other respondents, since software projects running beyond

---

58 While this claim should not be controversial, it is nonetheless limited to the context of Case A & B, and
to the last 10 year period (otherwise there is the assumption of a steady state of project turnover).
59 After a Wilcoxon rank sum test (W=117.5, p=0.14), we fail to reject the null hypothesis that there is no
difference between the distributions, at the 0.05 confidence level (evaluated by location shift).
10 years without progressing to ‘in-use’ are rare. Interestingly, there is support for this hypothesis (that the cause is that the respondents were early into their careers) in Case A, since there is a significant difference between the reported years of software engineering experience of the two respondent groups (Mean of 5 years vs. 15), but not for Case B (Mean of 17.5 vs. 16.25)\(^{60}\). Hence, it is possible that all of Case A’s respondents would have answered that either ‘Most’ or ‘Almost all’ of their projects had been in-use, if their experience been sufficiently long in duration to have witnessed their projects mature. The same cannot be said for Case B, leaving the less desirable (ii) & (iii) as the plausible reasons for the low ‘in use’ rate.

This revelation supports the researcher’s observations that more of Case A’s projects tended to be ‘problem-driven’ than Case B’s, at least in the sense that there was a time and business-critical need for the project, with management sponsorship and strong intention to commit to delivery. Indeed, in interviews, some of Case B’s practitioners described their projects as research or feasibility projects, in the sense that the customer had an interest in a new technology or a problem, but commitment to fund the project after the requirements had been specified (a pre-requisite for cost estimates) was not guaranteed. Hence, value-oriented RE in Case B is especially motivated by this question’s responses, since the value of these potential software projects is more frequently under question than in Case A.

**Takeaway Point #52: On average, ‘Most’ projects had progressed to ‘in-use’ in both Cases**

Thankfully most of the respondents’ project experience is suitable for the questions on the usefulness of implemented requirements. Interestingly, Case B had significantly higher incidence of projects that did not progress to ‘in-use’ than A, indicating that there is greater scope for the usefulness of value oriented decisions in Case A’s software projects.

### 6.1.6.3 For how many of these projects (from 6.1.6.2) do you have any knowledge of the usefulness or pertinence of the implemented requirements? (Case A: Q4.a, Case B: Q3.a)

It cannot be assumed that the respondents knew about the usefulness of implemented requirements in all of their projects that ever progressed to ‘in-use’, since some might not have been involved in phases with user involvement such as deployment or testing. Fortunately for this survey’s later questions, the combined response set’s median is ‘Most’ projects (for which the participant has knowledge of the usefulness of implemented requirements), with statistical significance\(^{61}\).

![Figure 6.4: Proportion of projects for which the respondent had knowledge of the usefulness or pertinence of implemented requirements — distribution of responses](image)

Case A’s mode and median response of ‘Almost all’ projects is perhaps not so surprising, given that the customers for Case A’s projects belong to the same organisation as the respondents. It should

\(^{60}\) A Welch two sample t-test on the ‘years of software development experience’ (Section 6.1.7.4) for the two respondent groups (p<0.05 indicates difference); Case A: t(8)=4.35, p=0.01; Case B: t(4.81) = -0.12, p=0.91

\(^{61}\) A Wilcoxon signed rank test supports the alternative hypothesis that the ‘true’ median response is greater than ‘Around half’ of the projects (had progressed to ‘in-use’): V=40, p=0.005
also be noted that all respondents reporting ‘Almost none’ or ‘Some’ projects, reported having ≤5 years software engineering experience, and so their projects were unlikely to be mature enough to enable such knowledge.

Conversely, such a positive result for Case B (having a median of ‘Most’ projects) might surprise the reader, given the often role-specific and compartmentalised nature of consulting work; One might assume that communication about a project with its stakeholders would end once the work had finished, but this appears not to be the case for the majority. A contributory factor might be the industry-wide popularisation of the ‘DevOps’ software delivery approach [Humb11], replacing ‘throwing the software over the wall’ delivery [Cohn05, p.23].

### 6.1.6.4 For how many of these projects (from 6.1.6.3) do you have any knowledge of the costliness of the implementations of the requirements? (Case A: Q4.a.i, Case B: Q3.a.i)

Across both cases, only approximately half of the respondents had knowledge of the costliness of the requirements’ implementations in ‘Most’ or ‘Almost all’ of the projects for which they knew of the usefulness of the implemented requirements. This deviation from the ‘high knowledge’ trend of the previous responses is surprising if we are to believe that Case A & B adopt best practice in cost estimation. (Practitioners are often required to estimate the cost of requirements in development effort [Cohn05, Somm11a], and checking past estimates against their realised values is critical for future accuracy [MoJø03].) Therefore, these results support claims in the literature that retrospective evaluation (or ‘post mortem’) of software is rarely performed as rigorously as it should be [Dist02].

---

**Point #53: The usefulness of requirements was, on average, known for ‘Most’ of the projects**

Despite the researcher’s initial concerns, the representative sample appears to be capable of answering questions on the usefulness of implemented requirements. Case B’s respondents (consulting) reported less knowledge about value than A’s.

---

62 The * symbol in the summary boxes should read: …of the projects that the participant was asked about.

63 Case A’s distribution does not sum to 100% because the remainder responded ‘Don’t Know or N/A’.

64 A Wilcoxon signed rank test supports the alternative hypothesis that the ‘true’ median response is greater than ‘Some’ of the projects (for which the respondent has knowledge of costliness): \( V=196, p=0.0003 \)
who doesn’t), hence indicating the presence of the common but less than ideal ‘stovepipe’ communication model [Chu04, p.66]. A plausible explanation is that the ‘Almost none—Some’ respondent group and the ‘Most—Almost all’ group are both populated by members having visibility of the opposite ‘sides of the wall’. In other words, it is hypothesised that the ‘Most—Almost all’ group has more business analysis experience than software development, and vice versa, (typically, project/business managers are responsible for cost tracking). The data from Case B’s responses supports this hypothesis, in that reported experience in ‘Business Case (Feasibility Analysis)’ (Q17.a) is positively correlated with knowledge of costliness (Q3.a.i) (Kendall’s rank correlation tau=0.42, p=0.06), while experience in ‘System (Technical) Testing’ (Q17.i) is negatively correlated with knowledge of costliness (tau=−0.6, p=0.001).

Interestingly, such a correlation does not exist in Case A’s responses. With insight from the researcher’s time spent with Case A, it is hypothesised that the lack of correlation is contributed to by a weaker partition between Case A’s respondents’ software engineering activity experience type (a pre-requisite for the aforementioned correlation). This insight is supported by the participant’s self-reported expertise: While in Case B, experience with the ‘User Requirements’ activity is negatively correlated with experience in ‘Component Construction (Programming)’ (Kendall’s rank correlation tau=−0.6, p=0.01), there is no such correlation in Case A (tau=−0.14, p=0.70).

**Point #54: The costliness of requirements was, on average, known for ~Half of the projects**

Findings on the incidence of wastefulness will be more robust than findings on its negative consequences, since the costs of requirements implementations are not widely known. Again, Case B’s respondents reported less value-oriented knowledge than A’s. (Perhaps related: Case B’s practitioners had stronger divide between the analyst/programmer roles.)

6.1.6.5 **How many of the projects (from 6.1.6.2) do you know of that achieved (at least) their intended benefits? (Case A: Q4.b, Case B: Q3.b)**

The participants were asked to report on the proportion of projects that achieved their intended benefits out of those that progressed to ‘in-use’. Hence enabling an understanding of the projects’ ultimate success in the context of requirements waste or inadequacy.

![Figure 6.6: Proportion of projects that achieved their intended benefits — distribution of responses](image)

In both cases the median response was ‘Most’ projects. The majority (true proportion >50%) of Case A’s population are highly likely to be aware of at least ‘Around half’ of their projects that achieved their intended benefits, as evaluated by a one-sample proportions test ($X^2(1)=5.8$, $p=0.008$; 95% CI: [0.62, 1]). The same cannot be said for Case B, whose true proportion of ‘Around half’ respondents is likely to be half the size of Case A’s ($X^2(1)=0.06$, $p=0.4$; 95% CI: [0.34, 1]), as indicated by Case B’s 34% lower bound in the 95% CI vs. Case A’s 62%.

---

65 Case A’s distribution does not sum to 100% because the remainder responded ‘Don’t Know or N/A’.
Case B has a higher incidence of benefit realisation failure than Case A. This will no doubt be influenced by Case B’s respondents having experienced, on average, twice as many projects (Section 6.1.6.1). Interestingly, Case B’s more experienced respondents (as determined by their number of projects) reported a higher proportion of projects that achieved their intended benefits\(^66\). This might have a simple explanation: it can take years for a software implementation to fully realise its potential benefits [Thor99], and the inexperienced respondents might not have seen them. However, the benefit realisation failures cannot be purely blamed on respondents with too little experience. Indeed this correlation is not sufficiently strong nor significant to be the only factor (Kendall’s rank correlation \(\tau=0.35\), \(p=0.10\)).

Aside from comparisons, the take away point is that neither organisation could claim no room for improvement in either realising or defining expected benefits, since there are so few responses to ‘Almost all’ projects. Statistically speaking, the combined response set supports the alternative hypothesis that the majority of Case A & B’s population know of less than ‘Almost all’ of their projects achieving their intended benefits (\(X^2(1)=17.9\), \(p=0.00001\)). Nevertheless, compared to earlier-discussed (Chapter 2.2) surveys in the literature (e.g., that 30% of IT projects deliver their expected benefits [Nels07]), both organisation’s projects seem more successful in realising benefits than is typical (given that both have medians higher than the comparable ‘Some’).

Finally, it is pertinent to remark on the conceptual ambiguity and political biases that this question’s responses were likely subject to, at least in interpreting what it means for a software project to have achieved its intended benefits. It is quite revealing that when asked about the proportion of projects that had quantified objectives (Section 6.1.6.18), 88% of the respondents answered between ‘Almost none’ and ‘Some’ of their projects in Case B, compared to 45% of the respondents in Case A; It is easy, and often favourable for businesses and their managers, to claim that intended benefits have been achieved when there is no clear criteria for it.

**Takeaway Point #55:** Both Case A & B have scope for improving software benefits realisation

It is almost certain that the majority of Case A & B’s population were involved in projects that failed to achieve their benefits. Twice as many of Case A’s population than B’s are likely to know of at least half of their projects achieving their benefits.

To conclude the discussion on these past four ‘how many projects…’ questions, Figure 6.7 visualises the gradual reduction of the respondents’ project set using a hierarchical bar chart, clarifying the constrained frame of reference that the respondents operated within while answering the later questions on those projects. (The four survey questions were presented to the participants in a similar hierarchical layout, in order to aide the comprehensibility of references to ‘of these projects’).

Especially evident from Figure 6.7 is the impact of Case B’s low rate of software project ‘in-use’ progression, meaning that Case A and Case B’s respondents effectively both have a similar number of projects for which they have knowledge of the implemented requirements’ usefulness, despite that Case B’s respondents have double the average number of projects experienced (Section 6.1.6.1).

\(^{66}\) A two-sample t-Test between project experience for the ‘Almost none’—‘Some’ group’s mean of 10 projects, and the ‘Around half’—‘Almost all’ group’s mean of 17.5 projects supports the alternative hypothesis that there is a difference between the groups: \(t(14)=-2.13, p=0.05\) [albeit only at \(p \leq 0.05\), and with the conversion of response intervals to lower bounds for the t-test – e.g., ≤15 was converted to 15].
6.1.6.6 On average over the projects you know about, how many of the implemented functional requirements (FRs) were redundant (i.e., not used)? (Case A: Q5, Case B: Q4)

The respondents were given six possible response options ranging from ‘Almost none of the FRs’ to ‘Almost all of the FRs’, including ‘Don’t Know or N/A’, while the wording and the ordinal spacing was kept consistent with the previous questions. The (statistically significant\(^{67}\)) median response in both Case A & B was ‘Some of the FRs’, as is visibly apparent in Figure 6.8.

---

67 A one-sample Wilcoxon signed rank test for the median response being greater than ‘None of the FRs’ supports the alternative hypothesis that the typical member of Case A & B’s population is aware of the existence of redundant implemented FRs (Case A: V=36, p=0.003; Case B: V=91, p=0.0003). (Even when the ‘Don’t Know or N/A’ responses are replaced with ‘Almost none’, rather than being removed.)
In addition, a one-sample proportions test supports the alternative hypothesis at the 0.05 confidence level, that the true proportion of Case A & B’s combined population knowing of at least ‘Some’ redundant implemented FRs is greater than 50%: X^2(1)=7.26, p=0.004 (95% confidence interval [0.6, 1]). In other words, in addition to the average member of Case A & B’s population knowing of redundant implemented FRs, the majority of the population likely does too. Hence providing evidence for this problem being real and current, at least in organisations sharing the context of either Case A or B.

However, the responses do not support the claims from the literature (Chapter 2) that on average, half of implemented FRs are redundant. This does not necessarily indicate disagreement, since the context of the projects surveyed here is focused. As such, it might be possible to find a similar result in the literature’s data if it were filtered to a similar context, e.g., bespoke vs. off-the-shelf, civil vs. non-military, etc. However, this is only conjecture, since as Jørgensen and Johnson’s discussion explicates, it is the norm to be unable to access industry survey data for commercial reasons [John06].

**Takeaway Point #56: The vast majority perceived ‘Some’ Implemented FRs to be redundant**

*It is almost certain that above ‘Almost none’ but below ‘Around half’ of all implemented functional software requirements were perceived to be redundant in Case A & B. Indicating the result’s robustness, Case A & B’s responses were highly similar. Therefore value based RE (and hence this thesis) is motivated, but claims that ‘half’ of FRs are redundant may be pessimistic.*

### 6.1.6.7 On average over the projects you know about, how many of the implemented non-functional requirements (NFRs) were wasteful or inadequate? (Case A: Q6, Case B: Q5)

There appears to have been less incidence of wasted or inadequate NFRs (Figure 6.9) compared to FRs, with considerably more respondents choosing ‘Almost none’ in both Case A and B. (One might not expect the responses to differ since FRs are ultimately derived from NFRs to achieve them [Paec04]. However, different response distributions can exist e.g., if the functions were useful, but their quality was wasteful or inadequate, since the question refers to NFRs as qualities of functions. An example FR and NFR was provided with the questions to encourage this mind-set (see in Appendix).)

![Figure 6.9: Proportion of the implemented NFRs that were wasteful or inadequate, on average over the projects — distribution of responses](image)

Indeed, it cannot be claimed with statistical significance that the majority of respondents were aware of the existence of wasteful or inadequate NFRs, as it could previously for FRs. (A one-sample proportions test does not support the alternative hypothesis that the true proportion of Case A & B’s combined population knowing of at least ‘Some’ wasteful/inadequate implemented NFRs is greater than 50%: X^2(1)=2.37, p=0.06, 95% CI=[0.49, 1]). Nevertheless, it is improbable

---

68 A one-sample proportions test strongly opposes the alternative hypothesis that the true proportion of the population knowing of > ‘Some’ redundant implemented FRs is greater than 50%: X^2(1)=21.3, p=1.
that repeated surveying would result in less than just under half (49%) of the respondents reporting knowledge of at least ‘Some’ wasteful/inadequate NFRs, given the 95% confidence interval, and the true median response being ‘Some’ in both cases.\(^6\)

Comparatively speaking, Case A’s proportion of responses greater than ‘Almost none’ is considerably lower than B’s (95% confidence intervals of: Case A: [0.28, 1] vs. Case B: [0.51, 1]), indicating that wasteful/inadequate NFRs are perceived to be a problem less frequently in Case A’s context (aerospace engineering) than in B’s (defence IT). Among many reasons, this perhaps due to the precision and safety engineering context embedded within Case A’s organisation, contributing to expectations for robust quality software.

Overall, there is no significant evidence to support the prioritisation of value-exploration effort for NFRs over FRs, despite that software is increasingly valued for its qualities rather than its functionalities [ReHB07]. However, lack of evidence for a problem is not necessarily evidence for a lack of the problem; Indeed, the finding should perhaps not be surprising, since:

1. While a software function can clearly be perceived as redundant or not (e.g., evaluated by the number of times used), wasteful implemented qualities (NFRs) of functions are less obvious, and are only perceivable as a problem to those who knew of the cost of implementing those qualities. E.g., a typical end-user is unlikely to be dissatisfied by a system response time being too quick or an analysis result being too precise (where slower or less precise would have equally sufficed). Furthermore, inadequate NFRs (e.g., too slow, precise, etc.) are likely to be perceived as poor software quality, rather than as poor RE, and if there are no costs involved, then NFRs specifying quality exceeding the need would not be problematic.

2. Knowledge of the costliness of implementing requirements was not common (as was established in the responses discussed in Section 6.1.6.4), which exacerbates the first point.

There is also doubt as to how the respondents would be able to evaluate whether an NFR was inadequate or wasteful, given that the modal and median response for ‘how often were requirements quantified (e.g., with fit criterion)’ (Section 6.1.6.19) was only ‘Some’ of the projects.

Point #57: Fewer implemented NFRs than FRs were perceived to be wasteful or inadequate

In contrast to FRs, there is insufficient evidence to claim that the majority of Case A & B’s population perceived at least ‘Some’ implemented NFRs to be wasteful/inadequate, despite that the majority of the sample (i.e., the survey respondents) did. Comparatively speaking, more in Case B (defence IT context) were aware of suboptimal NFRs than in Case A (aerospace context).

6.1.6.8 On average over the projects you know about, how costly were these requirements errors (the redundant/wasteful/inadequate FRs and NFRs)? (Case A: Q9, Case B: Q6)

It might be supposed that unused or wasteful implementations of software requirements are not a problem if their consequences are not costly. Conversely, it might be that a small number of wasted requirements cause a disproportionate amount of cost, e.g., in development effort, developer motivation, and so on. However, neither supposition is supported by the responses to this question,

---

\(^6\) Even when the ‘Don’t Know or N/A’ responses are replaced with ‘Almost none’, a Wilcoxon signed rank test supports the alternative hypothesis that the true median response is greater than ‘Almost none’ (Case A: V=21, p=0.013; Case B: V=78, p=0.0004).
whose median response was ‘Some’ of the project’s costs\(^70\), which is proportionally similar to the responses to ‘how many FRs’ and ‘how many NFRs’ ...were redundant? (Section 6.1.6.6 & 6.1.6.7).

However, there is the highest yet proportion of ‘Don’t Know or N/A’ responses. While this is unsurprising given the low response to the earlier question on the participants’ ‘knowledge about costliness’ (Section 6.1.6.4), it reduces the survey’s power to support claims about the majority. Indeed, in both cases, the proportion of responses other than ‘Don’t Know’ or ‘Almost none’ is insufficient to support the alternative hypothesis that the majority of the population (>50%) knew of the costs of wasteful requirements being greater than ‘Almost none’ (Case A: \(X^2(1)=1.45, p=0.11\); Case B: \(X^2(1)=0.56, p=0.23\)). Even when the response sets from both Case A & B are combined (having the primary effect of increasing the sample size rather than the proportion), the result remains borderline insignificant at the 0.05 confidence level (Case A & B: \(X^2(1)=2.37, p=0.06\)). To stress this marginal insignificance, it is unlikely that repeated surveying would find the proportion of >‘Almost None’ responses to be less than 49% (due to the 95% CI of [0.49,1]).

Swapping the burden of proof, we can confidently reject the alternative hypothesis that the majority of the respondents perceived the costs of wasteful requirements to be ‘Almost none’ of the projects’ costs (Case A: \(X^2(1)=9.09, p=0.09\); Case B: \(X^2(1)=10.56, p=0.09\)). Finally, to compare the two cases, a Wilcoxon rank sum test does not support rejection of the null hypothesis that there is no significant difference between the distributions of Case A and B (\(W=80.5, p=0.68\)), indicating robustness.

**Takeaway Point #58:** The cost of wasteful/inadequate FRs/NFRs is proportional to its incidence

There is strong evidence to support that wasteful/inadequate FRs/NFRs do have an impact on the total cost of the software project. That cost is proportional (‘Some’ wastage → ‘Some’ of the project costs). Case A & B’s responses appear to be similar.

**6.1.6.9 If these requirements errors (redundant/wasteful/inadequate FRs & NFRs) were costly, which were more costly?** (Case A: Q10, Case B: Q7)

The costliness of wasteful FRs may be different to that of wasteful/inadequate NFRs. (E.g., many NFRs concern architectural design options, which can be far costlier to change than independent functionalities if development has already started according to that architecture.) However, the responses shown in Figure 6.11 do not support such a hypothesis. Indeed, a one-sample proportions test for greater proportion than 50% of the participants responding with ‘FRs’ (the strongest response in both Cases for either of FRs or NFRs), does not support the alternative hypothesis that

\(^{70}\) In support of this median, is that a one-sample Wilcoxon signed rank test on Case A & B’s combined response set (‘Don’t Know’ replaced with ‘Almost none’, representing the worst case) supports the alternative hypothesis that the true median response was greater than ‘Almost none’ (\(V=171, p=0.00001\)).
the majority perceive FR errors to be more costly than NFRs, —and hence vice versa, (Case A: \(X^2(1)=0.27, p=0.3\), 95% CI=[0.14, 1]; Case B: \(X^2(1)=0.75, p=0.8\), 95% CI=[0.02, 1])\(^{71}\).

Figure 6.11: RE Errors that were most costly by requirement type — distribution of responses

It should be noted that the number of ‘FRs’ responses was double that of the ‘NFRs’ responses in both cases, but the previous paragraph’s conclusion is unaffected even when the ‘N/A’ and ‘\(\sim\)Equal’ responses are removed for the proportion test, due to their high proportion\(^{72}\). Hence, in support of the previous question’s findings on incidence (Section 6.1.6.7), there is no evidence to support the notion that value analysis should be focused on either NFRs or FRs in particular.

It is perplexing that 50% of the respondents answered ‘Don’t Know or N/A’ in Case B, given that only 31% of the respondents answered either ‘Don’t Know or N/A’ or ‘Almost none’ for the previous question on ‘how costly were the errors’ (Section 6.1.6.8)\(^{73}\). In the worst case, perhaps the inconsistent 19% changed their mind in that they actually did not know about the costliness of the requirements errors, (but if so, it is inconsequential for the previous conclusion to ‘how costly were the errors’\(^{74}\)). Otherwise, perhaps they were not aware of the breakdown of the costs by requirements type.

Finally, the respondents were asked “If you chose non-functional requirements, were they mostly: {Inadequate, Wasteful, Equally Split}” (Case A: Q10.a; Case B: Q7.a). In Case B, the respondents chose ‘Inadequate’, while in Case A, they were split between ‘Wasteful’ and ‘Equally Split’. The size of the ‘NFRs’ respondent group (previous question) is insufficient to make any robust conclusions, but the lack of a clear trend does support that there are no strong feelings amongst the participants.

**Takeaway Point #59: There is no evidence to support that either NFR or FR errors are more costly**

Twice the number of respondents considered redundant FRs to be more costly than wasteful/inadequate NFRs. However, the majority of respondents chose ‘N/A’ or ‘Equally Split’, causing statistical insignificance. Case A & B’s responses appear similar.

### 6.1.6.10 In your opinion, how many of these requirements errors could have been avoided (e.g., through better analysis or communication)? (Case A: Q7, Case B: Q8)

If redundant/wasteful/inadequate requirements are part of the essence of software engineering (e.g., in a develop-use-develop cycle designed to support ‘I’ll Know It When I See It’ RE [Boeh00]), rather than avoidable accidents (as in Brooks’ distinction [Broo87]), then an approach designed to avoid them would not be useful. Hence, the participants were asked for their perception on the proportion of

\(^{71}\) Both 95% CI’s are below the [0.51, 1] interval required for majority, which is most likely due to there being no clear majority for FRs or NFRs within Case B, and due to the small sample size in Case A.

\(^{72}\) When the ‘N/A’ and ‘\(\sim\)Equal’ responses are not considered, the sample sizes are too small for the 2:1 FR:NFR ratio to be significant (95% CI’s for the proportion of ‘FRs’: Case A: [0.28, 1] & Case B: [0.15, 1]).

\(^{73}\) This would not have occurred had the survey software been more powerful in its question routing.

\(^{74}\) As verified by re-running the statistical tests, after replacing the responses in (Section 6.1.6.8) with ‘Almost none’ for those who changed their mind in this question (Section 6.1.6.9)
redundant/wasteful/inadequate requirements (referred to as RE errors) that could have been avoided. As Figure 6.12 shows, very few respondents believed that none of the RE errors were avoidable.

Figure 6.12: Proportion of the wasteful implementations (requirements errors) that were considered to be potentially avoidable — distribution of responses

A one-sample proportions test supports the alternative hypothesis that the majority of the population (>50%) perceive ≥‘Some’ of the RE errors to be avoidable in the combined response set, and in Case B (Combined: X²(1)=7.26, p=0.004, 95% CI=[0.6, 1]; Case B: X²(1)=7.56, p=0.002, 95% CI=[0.65, 1]). The same cannot be said for Case A (X²(1)=0.36, p=0.27, 95% CI=[0.35, 1]), since despite that 64% of the responses were greater than ‘Almost none’, nearly a third were either unable or unwilling to speculate on the avoidability of the RE errors. If the former is true (unable to speculate), then post-mortem analysis and communication in Case A was inadequate, while if the latter is true (unwilling to speculate), as is most likely in such hypothetical questions [Offi12, p.5], then Case A’s respondents were essentially less willing (due to effort, politics, etc.) to rate the capability maturity, complexity or proneness to accident of their projects than Case B’s respondents were.

Overall, it is almost certain that the typical (median) population member (and likely the majority) in both cases perceives at least ‘Some’ of the wasteful/inadequate implementations to be avoidable. This, combined with the evidence that this thesis’ problem is perceived exists in practice (Section 6.1.6.6), is promising for the adoption and management/sponsorship of this thesis’ framework.

**Takeaway Point #60: Scope for the usefulness of value analysis in RE does exist**

In both cases the majority perceived at least some of the RE errors to be avoidable, showing that practitioners do consider there to be scope for improvement. Twice as many in Case B perceived the RE errors to be avoidable than A.

### 6.1.6.11 Briefly provide your opinion on why wastage occurred, if any did. (Case A: Q7.a, B: Q8.a)

A representative selection of the qualitative (freetext) responses are provided in Table 6.6. The themes running throughout the responses have been summarised in the ‘Theme Tags’ column.

---

75 The test’s insignificance is also partly due to Case A’s sample size, however, it would need to be quadrupled given the same distribution for a 95% CI greater than [0.5, 1], which was not viable with the resources.

76 A Wilcoxon signed rank test supports the alternative hypothesis that the true median response is greater than ‘Almost none’ [of the errors] (Case A: V=28, p=0.009; Case B: V=105, p=0.0004).

77 Stronger evidence for Case A’s majority can be found by changing the burden of proof, where a proportions test shows that it is highly unlikely that the majority of Case A’s population perceives ‘Almost none’ of the errors to be avoidable (X²(1)=5.81, p=0.009, 95% CI=[0.01, 1]).
Table 6.6: Respondents’ opinions on the causes of implemented FRs/NFRs being wasteful or inadequate

<table>
<thead>
<tr>
<th>In the respondent’s opinion, wasted FR/NFR implementations occurred because…</th>
<th>Errors</th>
<th>FR Errors</th>
<th>NFR Errors</th>
<th>Theme Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case A</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. “Engineers often do not have free rein over the solution space, implementing management-led solutions without full appreciation of the customers’ problems or alternatives.”</td>
<td>Don’t Know</td>
<td>Some</td>
<td>Some</td>
<td>limited-by-managers, poor req ts.-validation</td>
</tr>
<tr>
<td>2. “Requirements were not properly understood by the customer, or changed during development.”</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
<td>customer-didn’t-know, req ts.-changed</td>
</tr>
<tr>
<td>3. “Projects were not considered in context - focus was much more on specification of systems rather than what the systems had to achieve on behalf of its users and the business.”</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
<td>solutions-not-problems, poor req ts.-validation</td>
</tr>
<tr>
<td>4. “Lack of communication over requirements and how the capability should be used.”</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
<td>poor communication, solutions-not-problems</td>
</tr>
<tr>
<td>5. “There are two issues with NFR that draw attention: #1 - Immature ideas about adequacy of performance #2 - Architectural compromise. Both of these factors mean that it is difficult to get the NFRs right up front - it is quite reasonable in my experience to lay out a starting framework an then change more than half as the acceptability of certain compromises become clear.”</td>
<td>Around Half</td>
<td>Some</td>
<td>Most</td>
<td>customer-didn’t-know, quality-tradeoffs-unclear, req ts.-changed</td>
</tr>
<tr>
<td>6. “In our case, the customers of the software were in the same team as those developing it, so there could be many informal meetings to ensure that the appropriate functionality was produced.”</td>
<td>Around Half</td>
<td>None</td>
<td>None</td>
<td>question was misread, but is a valid perceived cause for the None’s ←</td>
</tr>
<tr>
<td><strong>Case B</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. “In my experience customers worry the systems will limit their ability, so feel a need to include any possible functionality. They also fail to account for any activities being outside of the system, delivered off line. The biggest client issue is defining process and method without testing the theory, a belief that they have created best practice on paper, then implemented it without real world testing; taking the implementation of software as an opportunity to rewrite the process. The best projects are those that take tried and tested best practice, and then automate and build on those processes and procedures within the system.”</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
<td>customer-wants-all out-of-scope→ignored, solutions-not-problems, poor req ts.-validation</td>
</tr>
<tr>
<td>8. “Poor requirements validation and/or justification.”</td>
<td>Some</td>
<td>Some</td>
<td>Some</td>
<td>poor req ts.-validation</td>
</tr>
<tr>
<td>9. “The importance and effort required to ensure adequate requirements elicitation and documentation has traditionally not been recognised in the programmes in which I have worked.”</td>
<td>Most</td>
<td>Some</td>
<td>None</td>
<td>poor req ts.-validation, RE-not-important</td>
</tr>
<tr>
<td>10. “Features were implemented assuming we knew what the customer wanted. Not enough time spent showing the customer the system.”</td>
<td>All</td>
<td>Half</td>
<td>Some</td>
<td>poor req ts.-validation, poor communication</td>
</tr>
<tr>
<td>11. “Most cost addition has been the result of poorly specified, or implied FRs. All of these could have been avoided through either better analysis or communication. Whilst I have seen an element of unused FRs and nonsensical NFRs, these have never been the biggest cost drivers.”</td>
<td>All</td>
<td>Some</td>
<td>Don’t Know</td>
<td>poor communication, customer-didn’t-know</td>
</tr>
<tr>
<td>12. “The end process was either not clear to the system developer, or the developer was working to a different spec to the one actually in use.”</td>
<td>All</td>
<td>Don’t Know</td>
<td>Most</td>
<td>poor req ts.-validation, req ts.-changed</td>
</tr>
</tbody>
</table>

78 To avoid ambiguity in the compressed headings, Table 6.6’s columns are mapped to questions as follows: 1st Column:(Section 6.1.6.11), 2nd:(Section 6.1.6.10), 3rd:(Section 6.1.6.6), 4th:(Section 6.1.6.7).

79 There are indeed many costs in a software project, but no claims about wasted requirements being the biggest were made by the researcher. Had this not been a research project guided by lack of knowledge rather than monetary gain, then it would have been guided by the problem’s monetary value.
Listed in order of occurrence, the themes are (the [A] & [B] tags represent their existence in Case A/B):

- **Poor Requirements Validation** (x7) [A][B] — indicates that requirements poorly aligned with the value proposition caused wasted development work, supporting the notion that initial requirements should not be treated as intrinsically valuable, unquestionable, nor immutable.

- **Solutions, not Problems** (x3) [A][B] — indicates that requirements describing the solution instead of the problem (a common occurrence [Berr09]) caused wasted development work.

- **Poor Communication** (x3) [A][B] — supports the notion that a key purpose of value analysis in RE should be to bridge the worlds of the developer and the users (language & motivation).

- **Customer Didn’t Know** (x3) [A][B] — indicates that RE cannot merely be asking stakeholders ‘what do you want?’, but that it requires proactive exploration of opportunity for value.

- **Requirements Changed** (x3) [A][B] — indicates that requirements were not ‘right first time’, (or what is ‘right’ changed), supporting the notion that confidence in req’s should be considered.

- **Limited by Managers** (x1) [A] — indicates that regardless of the Requirements Engineer’s (RE’s) awareness of sub-optimality, they felt bound to the prescribed solution. Where possible, communication of alternative means should be made, at least to reduce the RE’s liability.

- **Quality Trade-offs Unclear** (x1) [A] — indicates that wasteful or inadequate NFRs were implemented due to their effects on other qualities or architectural decisions being unclear. Hence, modelling (& learning from previous) causal relationships between NFRs is motivated.

- **Out-of-Scope → Ignored** (x1) [B] — indicates that requirements outside of the ‘software’ project’s scope were ignored, that were perhaps necessary for the software’s value to be realised.

- **Customer Wants All** (x1) [B] — indicates that the stakeholders did not reduce their initial requirements set (evaluated for value), presumably causing overruns/ suboptimal feature selection.

- **RE-not-important** (x1) [B] — indicates that wasted implementations were caused by general apathy for RE, most likely in favour of development work [BDFG05]. Informing the cynical stakeholders of the importance of RE with industry evidence would be useful here [DaCh06].

---

**Takeaway Point #61: The perceived causes of wasteful NFRs/FRs are targeted by value analysis**

The opinions of the participants on the causes of wasteful NFRs/FRs motivate modelling, communicating, and optimising the requirements for the software in the context of its expected benefits. Case A & B’s perceived causes were diverse, but a common set was found to exist. The top three causes were ‘Poor Validation,’ ‘Solutions not Problems,’ ‘Poor Communication’.

---

6.1.6.12 How frequently were requirements left unimplemented that would have been more valuable (better cost/benefit) than those implemented? (Case A: Q12.e, Case B: Q9)

Besides the existence of wasteful/inadequate implemented requirements, another symptom of RE value analysis failure is the existence of unimplemented but potentially more valuable requirements, since it is likely that their opportunity for development was consumed by developing less valuable requirements. As Figure 6.13 shows, the majority of the respondents do perceive such opportunity costs to occur. Indeed, a one-sample proportion test of the combined response set supports the alternative hypothesis that the majority of Case A & B’s population knows of at least ‘Some’ unimplemented but potentially more valuable requirements: $X^2(1)=3.7$, $p=0.03$. 

---
While Case A & B have a similar proportion of respondents knowing of at least ‘Some’ unimplemented-but-more-valuable requirements, Case A appears to have higher incidence of the opportunity costs. While this may indicate that Case A would benefit more from improvements to RE value analysis (especially in prioritisation and release planning), it might also indicate that Case A’s projects had more constrained timescales or bigger sets of requirements to consider for inclusion; Neither of which however detract from the former point on the usefulness of RE value analysis.

**Point #62: Requirements more valuable than those implemented were left unimplemented**

Opportunity costs of suboptimal RE value analysis are shown to exist, since most respondents had seen at least ‘Some’ projects having unimplemented-but-more-valuable requirements. Case A & B had a similar proportion of ≥ ‘Some’ projects.

**6.1.6.13 How frequently was it the case that…? (7 questions, Case A: Q12…, Case B: Q10…)**

The following seven questions investigated the occurrence of several ‘good practice’ indicators for RE value analysis, as well as obstacles to it. This is primarily to understand the gap between current practice, allowing for an assessment of the feasibility and magnitude of any change required for improvement.

The response distributions for the seven questions are shown in Figure 6.14 for comparability, and then each question is individual discussed (with its own distribution shown again).
You felt responsibility for the software’s value? (Case A: Q12.a, Case B: Q10.b)

End-users were easily contactable to explain reqt’s?

Business objectives for the software proj. were available?

Higher level bus. objectives (not for the software) available?

Business objectives were specified quantitatively?

Software/sys. req’s were specified quantitatively?

This question’s responses are clearly differentiated from the others in this survey, with at least half of the participants responding with ‘Almost all’ of the projects. The respondents generally felt as though they were responsible for the value of their software projects, with only 11% of the combined response set reporting to have never felt responsible for value\(^{80}\). This deviates from popular opinion in the academic literature that in the typical software engineering context, software engineers do not feel responsibility for delivering value (instead feeling responsibility for schedule and scope [Jain07]), which is blamed as a significant cause of software value failure [Boeh05]. While this is an indication that both Case A & B’s organisations are of a higher maturity than typically assumed by such literature, it cannot be ignored that in both organisations, plausible motivation exists for the participants to report that they felt responsible for value\(^{81}\). Nevertheless, the result is positive for motivating work

\(^{80}\) Of those who volunteered for a follow-up interview (Case B only), the perceived cause was feeling as though their job was strictly to specify or implement ‘software capability X’ without other concern.

\(^{81}\) Despite the survey being anonymous, the aggregated results would eventually be seen by the managers.
on value oriented RE, given that the opposite result would suggest that it would be challenging to persuade practitioners to adopt value oriented RE methods and tools.

The notion that responsibility for value improves value realisation is supported by correlations. A strong positive correlation exists between the number of projects that achieved their intended benefits (Section 6.1.6.5) and the number of projects that the respondent felt responsibility for providing value (Kendall’s rank correlation: Case A: \( \tau = 0.42 \), \( p = 0.14 \); Case B: \( \tau = 0.4 \), \( p = 0.09 \); Combined: \( \tau = 0.44 \), \( p = 0.01 \)). Value for responsibility is also negatively correlated with incidence of wasteful/inadequate NFRs in Case B (Q10.b, Kendall’s rank correlation \( \tau = -0.5 \), \( p = 0.04 \)). It would not be surprising if this correlation was causal given its reasonability, regardless of there being no statistically significant correlation for Case A’s responses (Kendall’s \( \tau = 0.09 \), \( p = 0.77 \)).

**Takeaway Point #63: Responsibility for value may be necessary, but is not sufficient for value**

Despite a significant correlation between benefits realisation & responsibility for value, value failures prevailed. The majority did feel responsibility, indicating that both Cases should be interested in the application of this thesis’ framework.

### 6.1.6.15 How frequently were the system’s end-users easily contactable to clarify or explain their requirements? (Case A: Q12.c, Case B: Q10.a)

Figure 6.16: Proportion of the projects where the system’s end-users were easily contactable to clarify or explain their requirements — distribution of responses

The majority of Case A’s respondents claimed that end users were easily contactable in ≥ ‘Most’ of their projects, whereas the majority of Case B’s respondents claimed the opposite (≤ ‘Some’). Given that Case A’s end-users belong to the same organisation and Case B’s context of work is consulting, this is likely to be due to differences in locations, commercial goals, culture, privacy, and so on.

Given the previously found difference between Case A & B’s software project benefits realisation rates (Case A’s true median of ‘Most’ vs. Case B’s ‘Around Half’ of the projects in Section 6.1.6.5), it is not surprising to see the difference between the cases’ end-user-communicability. Communication between developers and end-users is known to be important for a software project’s success [AbPa13], and is also apparent from Case A’s response [6] in Table 6.6. Nevertheless, there is more to realising benefits (and avoiding wasteful/inadequate requirements) than communicability, as supported by the statistically insignificant correlation in the combined response set (rank correlation \( \tau = -0.13 \), \( p = 0.42 \)).

Follow up interviews with Case B’s negative responders indicated that communication chains were often elongated due to subcontracting (where the software project’s end users were customers of Case B’s customers), and sometimes intentionally prohibitive due to cross-organisational security. Constraining the developer’s view of the problem space to the solution space through lack of, or prohibition of communication, cannot be good for the optimality nor pertinence of their solutions. Whether this is sufficient motivation for the stakeholders of future projects to change, depends on the magnitude of the assumed costs as well as organisational policy, but it should be kept in mind that most value oriented RE information would, in reality, not pose significant risk. A role based security model defining access levels might be useful, e.g., preventing ‘Programmers’ from viewing as-is capability statistics, instead showing relative (%) prescribed targets for the new system.
Overall, the responses indicate that significant improvements in Case A or B’s RE processes cannot be caused by the development organisation alone, requiring commitment from other stakeholders to provide and encourage communication about value concerns. The indicated difficulty in contacting end-users also suggests that applying Agile RE approaches would be challenging [CaRa08], and that its more ad hoc style of communication might cause project time delays, or worse, assumptions in its place. (In some contexts a viable strategy for reducing the incidence and consequences of wasteful/inadequate implemented requirements is the rapid release to users for quick feedback. However, this appears to be somewhat prohibited by poor communicability in a significant proportion of Case A & B’s projects, —as is also hinted at by the qualitative response [10] in Table 6.6).

Hence it becomes desirable to produce heavier weight RE models to attain ‘line of sight’ [SiWo09] between software requirements and their value proposition, such that the infrequent communications that do occur, are less likely to result in scrap or re-work caused by incorrect understanding, which would be more expensive due to longer communication delays. (To be clear, pure ‘Big Requirements Up Front’ [Amb12] (Waterfall style) is not advocated due to the existence of IKIWISI (I’ll Know It When I See It) requirements [Boeh00], and that all models will be ‘wrong’ to some degree compared to reality [Box87]; User feedback from frequent software releases cannot be substituted.)

**Takeaway Point #64: End user communicability may be necessary, but is not sufficient for value**

Despite Case A’s higher end-user communicability and higher benefit realisation than Case B, no statistically significant correlation between them was found. In both cases, the need for non-ad-hoc RE (especially needs elicitation) is supported.

### 6.1.6.16 How frequently were the system-owner's business objectives for the project available? (Case A: Q12.g, Case B: Q10.c)

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>50%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.17: Proportion of the projects where the system-owner's business objectives for the project were available — distribution of responses**

The responses show that the widely espoused best practice of specifying business objectives for software projects [Ieee11a, Ieee98, RoRo12b] is adhered to in a non-trivial proportion of industry projects. Indeed, the majority of the respondents (Case A: 54%, Case B: 62%) were aware of at least half of their projects that had done so. However, given the ubiquity and reasonability of this best practice, it is disappointing that this majority claim is not statistically significant for the populations. (A one-sample proportions test does not support the alternative hypothesis that the majority of Case A & B’s population (>50%) were aware of at least half of their projects having business objectives available for the software project: (Case A: $X^2(1) = 0, p=0.5$, 95% CI: [0.28, 1], Case B: $X^2(1) = 0.56, p=0.22$, 95% CI: [0.39, 1], Combined set (A & B): $X^2(1) = 0.59, p=0.22$, 95% CI: [0.42, 1]).

A strikingly large 45% and 38% of Case A & B’s respondents (respectively) were only aware of only ‘Some’ projects having the system-owner’s business objectives available. In those instances, the expected benefits to the business of those software projects were either not prescribed or communicated. This could be for a myriad of reasons, e.g., perhaps the projects were not considered large enough to justify the overhead, the respondents were not privy to them (where objectives are shared on a ‘need to know’ basis), or the system owners cared only for the prescribed technical requirements. Regardless, we cannot reject the hypothesis that most of the surveyed software
engineering practitioners were, in effect, left to guess or infer by themselves how the software requirements were instrumentally valuable for the software system owners.

**Takeaway Point #65: Business objectives in software projects are not as common as expected**

Despite being a component of mainstream RE specification standards, business objectives are not ubiquitous. Since this practice is similar to a basic component of this thesis’ proposed framework, it indicates potential difficulty in its adoption.

6.1.6.17 How frequently were the system-owner’s higher-level business objectives (i.e., not specific to the project) available? (Case A: Q12.i, Case B: Q10.d)

This question refers to the availability of the system owner’s higher level business objectives that are not specific to the software project, e.g., a business unit’s objectives that (should) guide many projects. If these objectives are made available, then the software project’s business objectives can be placed into the context of the system owner’s desires. In other words, in a chain of value derivation $X \rightarrow Y \rightarrow Z$, $X$ could be software system requirements, $Y$ software project business objectives, and $Z$ this question’s ‘higher-level’ business objectives.

The responses indicate that provision of these higher level business objectives is significantly rarer than the previously discussed software project business objectives; The 95% confidence interval for the proportion of responses greater than ‘Some’ projects is a mere [0.19, 1], —less than half of the previous question’s [0.42, 1]. Furthermore, a mere 22% of the combined response set reported both >’Some’ projects with software project business objectives and >’Some’ projects with higher level business objectives. These results are not surprising, given that higher level business objectives are far removed from the concerns that a typical software developer or even requirements engineer would typically be held responsible for. This is despite that, amongst numerous benefits (discussed in Chapter 2), they would contribute to the description of the ‘true’ underlying problem to be solved, and when defined at appropriate levels of abstraction, enable ‘line of sight’ to the system owner’s strategy, —which has the capability to improve innovation, increase motivation [SiWo09], expose assumptions, and improve the pertinence & optimality of solutions [Lams09a].

Finally, approaches proposed in the literature designed for modelling the strategic alignment of IT (see Chapter 4.6) are often presupposed on the assumption that these objectives will be available, but as the responses show, this is an idealistic assumption. Indeed, in follow up interviews, and in the researcher’s experience at the organisations, objectives at higher levels than the department responsible for the software project were inaccessible to them or vague, due to confidentiality.

Comparatively speaking, and in alignment with this researcher’s expectations, Case A has higher incidence of ‘system-owner non-project-specific business objective availability’ than B, likely due to B’s consulting context where privacy concerns and ‘need to know’ policies are more prohibitive.

**Takeaway Point #66: Higher level Business Objectives (BOs) are ~2x as rare as software project BO’s**

It is not the norm for a software project’s practitioners to know what the system owners desire outside of the software project’s scope. So, being able to answer more than one level of ‘why?’ questioning above requirements would be a change.
6.1.6.18 How frequently were the system-owner’s business objectives quantitatively defined? (Case A: Q12.h, Case B: Q10.e)

The majority of the respondents perceived quantification of business objectives to be uncommon\(^{82}\). This question received the strongest ‘Almost none’ response of all the survey’s questions. Furthermore, a mere 18% of the combined response set reported both >‘Some’ projects with software project business objectives and >‘Some’ projects with quantitatively specified objectives. This essentially means that in most projects the stakeholders could declare success subjectively and qualitatively regardless of extent or magnitude, (e.g., ‘did you feel as though costs were sufficiently reduced?’). In follow up interviews with the participants, it became clear that a software project’s success was often declared with retrospective claims of benefits, independent from the original objectives (and hence independent from quantified targets). More importantly, it indicates that the software requirements were not evaluated before implementation (or selection of alternatives) for whether they would be likely to achieve some degree of benefit (as defined by the objectives), since ‘some degree’ was undefined\(^{83}\). Hence, uncertainty about the degree of the requirements’ sufficiency was ignored, which has implications wider than just value (e.g., opportunity for risk identification).

While the aforementioned proportions test supports the ‘majority had unquantified objectives’ claim for the combined response set as well as for Case B\(^{84}\), the same cannot be said for Case A. Indeed, nearly four times as many of A’s respondents than Case B reported ≥‘Around half’ such projects. While this is indicative of higher process maturity (as in CMMI [Sei12]), it could merely be that the quantifications were removed before Case B’s practitioners were given project documentation, (given that the ‘need to know’ constraint is tighter in consulting than in a one-organisation context).

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Proportion of the projects where the system-owner's business objectives were quantitatively defined — distribution of responses}
\end{figure}

\textbf{Takeaway Point #67: Knowing how much value a software project should produce is not common}

Being able to unambiguously decide whether a requirements set will produce, (and then if a project has produced) ‘enough’ value, —and so to track any improvement, seems like an obvious (necessary) improvement to increase value realisation rates.

\(^{82}\) A one-sample proportions test for the majority (>50%) of Case A & B’s population knowing of ≤‘Some’ projects with quant. objectives supports the conclusion (Combined set: \(X^2(1)=5.33, p=0.01, 95\% \ CI: [0.57, 1]\)).

\(^{83}\) Such apathy for quantification is less likely to be found for more traditional (cost(rather(than(benefit oriented, or ‘iron triangle’ [LiPe01]) project planning concerns, such as resource allocation or scheduling; A project is unlikely to progress in an organisation without, e.g., an estimate of the number of FTEs required for it. Hence, if tradition changes away from cost-oriented success indicators, this practice may become more popular.

\(^{84}\) Case A: \(X^2(1)=0, p=0.5, 95\% \ CI: [0.28, 1]\). Case B: \(X^2(1)=7.56, p=0.003, 95\% \ CI: [0.65, 1]\).
6.1.6.19 How frequently were the software requirements quantitatively defined (e.g., with fit criterion)? (Case A: Q11.a, Case B: Q10.f)

The majority of the respondents perceived quantitative specification of requirements to be uncommon. Indeed, we cannot reject the null hypothesis that the minority of Case A & B’s population know of more than ‘Some’ projects with quantitative requirements (one-sample proportion test for the alternative hypothesis of a >50% proportion: combined response set $X^2(1)=2.37$, $p=0.93$). This is despite that the ‘quantify requirements’ best practice is widely espoused [GiCo08, Ieee11a, Ieee98, Maid06, RoRo12b], indicating that in current practice, there is rarely rigour in determining the suitability of software solutions for meeting the requirements. Essentially (or cynically), a great deal of time and effort appears to have been wasted by prescribing likely-incomplete accounts of software quality taxonomies [Iso01] masqueraded as requirements, e.g., ‘system should be usable, fast, secure’, etc. (If not quantified, they would be applicable to most software projects but would be useful in very few for purposes other than for ‘ticking the box’ — i.e., to say that NFRs have been specified and hence process has been complied with).

Follow up interviews revealed several potential causes for this demonstrated aversion to quantification, where a common theme was the notion that some merely dislike quantification. In the experience of the interviewees, most are simply comfortable with the ease of definition and the freedom (or range of interpretations or ‘get out clauses’) associated with qualitative requirements. Indeed, the probability of satisfying a typical qualitative requirement is inordinately higher than satisfying a quantitative requirement, causing less political and personal obligation. Furthermore, it would not be out of the ordinary to not have quantitative requirements. Hence if they don’t do it then there is low risk of being the exception and being blamed for not doing so, but if they do, then they introduce the risk of failing to achieve the requirements, or worse, exposing that the software [the work of their colleagues] was not as satisfactory as was planned. In the words of Upton Sinclair, “It is difficult to get a man to understand something, when his salary depends upon his not understanding it” [Sinc95].

Another concern held by the interviewees was that prescribing ‘wrong’ quantitative targets could cause unnecessary expense or inadequacy, given that at the RE phase of a project, perfect knowledge about what will be optimal post-implementation does not exist. However, this concern is partly based on the common misconception that quantification and measurement are exclusively for things we can be certain about, rather than to make explicit and gradually reduce uncertainty about them [Hubb10]. Finally, some interviewees claimed that specifying requirements qualitatively was favoured because there was insufficient time for RE, and finally, sometimes due to ignorance of best practice.

Unfortunately, changing this practice faces psychological as well as socio-political barriers; People are averse to uncertainty (often irrationally) [Epst99], and also to being ‘wrong’ about quantitative requirements, since they are likely to be more penalised than if they were ignorant of the ‘right’ numerical target (where satisfaction of a requirement could be claimed regardless). This finding is not unique to Case A or B. For example, a recent survey found that 85% of its respondents claimed
that all NFRs had been satisfied by the end of the project, but all respondents except one were vague in describing how the NFRs were validated, implying that they were not [FAAC14].

**Point #68: Quantifying requirements might be best practice, but it is far from common practice**

Most often, there was no clear cut criteria for deciding whether a designed software capability or quality was sufficient. This along with the previous response indicates that the proposed framework’s quantitative aspects will be unfamiliar.

6.1.6.20 How frequently were software requirements explicitly traced (i.e., relationships made) to their proposed benefits (e.g., to business objectives)? (Case A: Q12.j, Case B: 10.g)

The next four questions investigated the occurrence of the essence of modelling the instrumental value of software requirements in an organisational context, i.e., requirements→benefits traceability—a basic method for it by itself (as reviewed in Chapter 4.4.1). The following responses contribute to understanding the magnitude of change required to implement this thesis’ proposed framework.

The participants were first asked to recall the proportion of their projects that had explicitly traced (i.e., documented, modelled, etc.) the software requirements to their expected benefits in the pre-implementation phases of their software projects. As Figure 6.21 shows, both response distributions are skewed toward ‘Almost none’ of the projects, indicating that it is not common practice.

![Figure 6.21: Proportion of the respondents’ projects that had explicitly traced software requirements to their proposed benefits — distribution of responses](image)

Indeed, the majority of Case A & B’s combined population are highly likely to have seen requirements→benefits traceability in only ≤ ‘Some’ of their projects (supported by a one-sample proportions test: \(X^2(1)=9.48, p=0.001, 95\% \text{ confidence interval}=\{0.65, 1\}\)). This again supports the notion that in industry, requirements are often treated as intrinsically valuable. Finally, both distributions are highly similar\(^{85}\), indicating that the practice’s scarcity is not specific to either context.

Follow up interviews in both cases revealed that traceability matrices were almost exclusively used, despite that other techniques (e.g., directed graphs) may often more suitable (as studied in [LiMa12]). Amongst the particular matrix techniques, the House of Quality (Chapter 4.4.2) was popular in both cases, but B’s standards and best practices encouraged the use of MODAF SV-5 (Chapter 4.4.1), (also referred to as a ‘Requirements Traceability Matrix’ [IEEE11a]). Examination of requirements documents at Case A & B supported this trend, as well as frequencies of occurrence in Figure 6.21.

---

\(^{85}\) A Wilcoxon rank sum test leads us to be unable to reject the null hypothesis that both ‘true’ distributions (Case A’s & B’s) are the same, as evaluated by location shift (\(W=86, p=0.94\)).
It was apparent from the interviews that frequently, only ‘top level’ software requirements were traced to business objectives, as opposed to refinements or decompositions at a more technical level. Various reasons claimed for this included that:

1. It was considered to be sufficient for satisfying the stakeholders that the business objectives would be achieved by the traced software capabilities (for rudimentary requirements validation).
2. Tracing to lower level requirements would face consequences of diminishing returns, since while not representing the mechanistic details of potential for value, the top level requirements were considered a sufficient ‘hook’ into those details for encapsulating the lower levels. One participant claimed that “in the real world”, using rigor on principle tends only to add costs.
3. Expanding the traceability matrix to include the lower level requirements would be unmanageable for scalability of effort, and the typical A4 paper requirements document format.

The pros and cons of these viewpoints have already been thoroughly discussed (namely in Chapter 2.1.2.3 and throughout Chapter 5), but before conclusion, two opposing arguments must be balanced:

1. It is often implied in the literature that if practitioners do not use a particular methodology, then they are declining to use methodological or rigorous practice [Petr13]. However, most practitioners, especially those with enough experience to be included in the representative samples, will have previously exhibited competence at deploying particular tools to solve problems. Indeed, other studies show that practitioners who decline to use specific tools nonetheless have thoughtful and systematic practices for their projects’ contexts [Petr09].
2. As this survey makes evident, most software projects exhibit value realisation failures that were deemed to be avoidable retrospectively. In other words, current practice is not perfect, and so neither are the practitioners’ rationalisations for it.

### Point #69: Most projects did not have explicit software requirements → benefits traceability

For reasons explored in the next question, it appears as though most of Case A & B’s population have rarely seen traceability between requirements and their expected benefits. This means that this thesis’ framework will be unfamiliar in practice.

#### 6.1.6.21 If traceability from requirements to benefits/objectives was made available, was it… (or if it wasn’t, do you think it would have been…): (Case A: Q13, Case B: Q11)

The next three questions aimed to understand the usability/utility of the aforementioned practice.

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>50%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>18%</td>
<td>18%</td>
<td>55%</td>
</tr>
<tr>
<td>Case B</td>
<td>9%</td>
<td>31%</td>
<td>44%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>50%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>9%</td>
<td>18%</td>
<td>45%</td>
</tr>
<tr>
<td>Case B</td>
<td>6%</td>
<td>6%</td>
<td>25%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0%</th>
<th>50%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A</td>
<td>9%</td>
<td>64%</td>
<td>9%</td>
</tr>
<tr>
<td>Case B</td>
<td>13%</td>
<td>50%</td>
<td>19%</td>
</tr>
</tbody>
</table>

Figure 6.22: Proportion of the projects that match the statements — distribution of responses
The majority of Case A & B’s respondents did or would consider ‘requirements → benefits traceability’ to be useful in ≥ ‘Most’ projects (significant for the majority of A & B’s combined population: \(X^2(1)=5.33, p=0.01, 95\% \text{ CI}=[0.57, 1]\)). However, the same cannot be said for the respondents’ opinions on its ease of comprehension (combined set: \(X^2(1)=1.33, p=0.12, 95\% \text{ CI}=[0.45, 1]\)), nor on its ease of creation (combined set: \(X^2(1)=0, p=0.5, 95\% \text{ CI}=[0.32, 1]\)). This trend suggests that if such traceability was not made explicit, it was more often due to its perceived disadvantages rather than its advantages. Hence, if this thesis’ framework is to be implemented in an organisation, its usability features (e.g., automated visualisation, guided benefit elicitation, tool supported re-use etc.) need to be advertised.

In follow up interviews, the participants primarily referred to difficulties on the themes of:

- Non-mechanistic traceability encouraged by the use of matrices (having only two dimensions to co-relate); While the matrix showed that a requirement would lead to a benefit, it was often not clear how, in which scenarios (or use cases), nor to what extent (sufficiency/necessity).
- High availability of suggested causes (technical solutions), but not of expected effects (anticipated benefits). Hence creating such traceability would require more communication and sharing of mental models, which is not necessarily negative, but is often limited by resources or confidentiality.

The response distributions were similar between Case A & B for the perceptions of the usefulness of such traceability. However, strong differences exist in the perceptions of its ease of comprehension and especially its ease of creation, with Case A’s respondents being more negative than B’s. This is not surprising, given that Case A had ‘better’ responses to the previous questions on related factors, such the availability of business objectives, end-user-contact-ability, and so on. Perhaps Case B’s respondents merely preferred the task, but it may also indicate that Case A’s requirements sets are more complex (in size, dependencies, levels of hierarchy, ‘line of sight’ to benefits, etc.) than B’s.

Finally, two respondent groups can be distinguished, depending on their response to the previous question on the incidence of ‘requirements → benefits traceability’ (Case A: Q12.j, Case B: 10.g): One set of respondents recalled the usefulness/utility of previous implementations of the traceability, while the other speculated on its hypothetical usefulness/utility if it were implemented. As Figure 6.23 shows, there are (statistically significant\(^{87}\)) differences in perceived usefulness/utility between these groups. In summary, those who experienced requirements → benefits traceability perceived it to be more useful and usable than those who had not experienced it. This is perhaps obvious, given that rational people generally tend to do things that they perceive to be useful and usable, and vice versa. However, it does not fully explain the observation, since there was not an exact correlation between the two\(^{88}\), and the participants would not likely have had total autonomy over their actions (in industry, autonomy is often constrained by roles, responsibilities, management approval, and time constraints). Regardless of its cause, it is encouraging for this thesis’ framework that those with experience have the most positive perceptions (rather than those with only word of mouth or thought experiments), since they will be the better informed and hence more credible group. Finally, given the trend, it cannot be rejected that those with no experience of such traceability would have better

---

\(^{86}\) Case A & B’s differences might also have been (but were not) due to a difference in the proportion of respondents speculating about the usefulness/usability, as opposed to those recalling actual experience.

\(^{87}\) A Wilcoxon rank sum test supports the alternative hypothesis that the distributions of responses are different between the two groups’ (‘recalling’ vs. ‘speculating’) opinions on the usefulness of requirements → benefits traceability: \(W=129, p=0.04\), (Medians=\{'Recalling': Almost all, ‘Speculating’: Most projects\}).

\(^{88}\) Frequency of ‘requirements → benefits traceability’ vs. perceived usefulness: Kendall’s \(\tau=0.24\), \(p=0.16\)
opinions of it, after experiencing it. Indeed, this is the rationale behind free software trials (inspiring the currently popular ‘freemium’ software business model [Seuf14, p.2]).

### 6.1.6.22 How frequently was it the case that…: (Case A only: Q12.b, d, f; Q11.b)

This final series of questions concludes this chapter’s discussion of non-demographic-oriented questions. (This subsection’s following four questions were asked in Case A’s survey, but not in Case B⁹⁹, and so serve only to provide extra context about the requirements’ engineering practice in Case A.)

Firstly, the participants were asked how frequently they had “an understanding of how much business value the software requirements would create (i.e., ‘line of sight’)” (Q12.b).

![Figure 6.24: Proportion of the projects where the respondent had an understanding of how much business value the software requirements would create — distribution of responses](image)

The majority of respondents (72%)⁹⁹ reported having such line of sight in ≥ ‘Most’ of their projects (Figure 6.24), which is surprising. Given the low incidence of explicit ‘requirements→benefits traceability’ (Q12.j) (as well as low availability of business objectives, and rare quantification of requirements and objectives), this reported line of sight will in most cases, have come from the respondents’ uncodified (neither documented nor communicated) mental models of requirements-benefits traceability⁹¹. This is encouraging for the usability of this thesis’ framework, since it indicates that practitioners already construct at least parts of the needed models in their minds, but are not exploiting them to their full advantage via codification, communication, and re-use.

---

⁹⁹ They were not apparent when creating the first survey distributed to Case B, but the researcher’s curiosity on the matters lead to their inclusion in the second. This has no impact on this thesis’ research questions.

⁹⁰ Not significant for claims about the majority of the population: X²(1)=1.45, p=0.11, 95% CI: [0.44, 1].

⁹¹ Indeed, those who reported having line of sight in ‘Almost all’ of their projects did not report different (i.e., higher) incidence of explicit ‘requirements→benefits traceability’ (Wilcoxon rank sum W=18.5, p=0.3).
More crucially, those who reported having line of sight in ‘Almost all’ of their projects did not report significantly different incidence of software project benefit realisation ($W=28$, $p=1$), nor redundant FRs ($W=20$, $p=0.24$) or NFRs ($W=36.5$, $p=0.3$). Therefore, while one might conclude that having this line of sight provided no obvious ultimate advantage, it might actually be more correct to say that the practitioners’ line of sight was either overly confident or otherwise incorrect, given the previously reported benefit realisation and wasteful/inadequate requirements rates. This further motivates this thesis’ framework, given that it allows for making confidence explicit, learning from past projects (hence allowing confidence calibration), and refutation of incorrect assumptions by communication of the value mechanisms that the software’s value would depend on.

**Takeaway Point #71: Practitioners are likely to be able to construct explicit ‘line of sight’ models**

Although most projects did not have explicit requirements-benefits traceability, the majority of Case A’s respondents believed they had personal understanding (i.e., mental models) of how much value the requirements would lead to. Hence, this thesis’ framework externalisation and codification of those models should not be overly challenging for Case A.

The last three non-demographic questions asked in Case A explored the incidence of consequences or inadequacies of poor RE. The participants were asked “How frequently was it the case that”:

- Q12.d: “Software project releases were late due to delivering functionality not considered to be essential to the end-user or business”;
- Q12.f: “The requirements did not express the true needs of the end-users or business (i.e., they were unhappy despite the software meeting the requirements)”;
- Q11.b: “Requirements tended to prescribe technical solutions rather than describe problems”.

As Figure 6.25 shows, the majority of the respondents had seen all of these phenomena occur in at least ‘Some’ of their projects. The most prevalent bad practice is the widely reported (and old [SBJA98]) convention of designing the solution without adequately exploring the problem (Q11.b). This thesis’ framework directly addresses this and Q12.f’s problem, and indirectly Q12.d’s. Indeed, interviews with the practitioners indicated that their software projects sometimes competed against other software projects (internal and external) that could replace them, and to still be pertinent in current business conditions. Hence, being able to ascertain which software requirements provide the essential value to the business, for release planning, is a desirable capability of the framework.

**Takeaway Point #72: Insufficient value analysis in RE has led to late projects and dissatisfaction**

Further motivation for this thesis’ framework is Case A’s incidence of late projects, and ‘true needs—requirements’ mismatch.

### 6.1.7 Demographics & Context for the Questionnaire’s Respondents

The following questions were dedicated to eliciting information about the participants’ experience, in order to understand the respondents’ characteristics, and hence to improve the external validity of the survey. I.e., the results obtained from this survey can be better compared to other surveys, by considering the similarity of the respondents’ characteristics. Furthermore, eliciting the
respondents’ amount and type of software engineering experience, enables examination of hypothesis such as: is it the case that those with no RE experience in their career (e.g., developers who convert specifications to code) would find ‘software requirements→benefits traceability’ less useful than those with more RE experience. Such hypothesis are tested by examining the correlations (with significance tests) between all of the questionnaire responses, as previously presented in Figure 6.1.

6.1.7.1 Describe your involvement in the following Requirements Engineering activities in the last 10 years. (Case A: Q15, Case B: Q18) Options: {None, Some, Substantial} experience

The participants were provided with a full list of the requirements engineering activities, as defined by Cleland-Huang [Clel05, fig.1], and were asked to rate their experience with them. To avoid survey measurement error caused by misunderstanding, the activities were briefly described with the question.

As Figure 6.27 shows, the most popular activity was ‘Requirements Specification’, having 48% of the response ratings as ‘Substantial’ [experience]. The least popular were ‘Requirements Analysis’ (30%) and ‘Requirements Verification & Validation’ (33%), compared to the mean of 36% (std. dev.=0.07%). Within this sample of experienced RE practitioners, it appears that activities consisting of comprehending and analysing requirements are rarer than writing them (contrary to programming where code is read far more than it is written [Chen07]). In the words of the Cleland-Huang’s RE activity definitions, the participants were more experienced in “describing [the system to be developed]”, than in “gaining understanding of the product to be developed” and assessing whether “the specified requirements may [have] fallen short of capturing the real needs” [Clel05].

This is further indication of the recurring trend for treating requirements as intrinsically valuable and primarily as solution designs. It also adds support to the qualitative responses in Section 6.1.6.11, which conclude that the most popularly perceived cause of redundant/inadequate requirements is lack of requirements validation, which is a primary purpose of this thesis’ framework.

Figure 6.26: Experience in the requirements engineering activities — distribution of responses. (In Case A, this question was made optional due to time constraints on the practitioners. Hence, while Case A’s distributions sum to 100%, they represent 63% of Case A’s participants).
Takeaway Point #73: Requirements Analysis and Validation are less popular than Specification

This supports the participants' views that lack of validation is the most common reason for FR/NFR redundancy, and hence the framework. It also means that the priorities attached to the RE activities must change in Case A & B, for the framework to be used.

6.1.7.2 Describe your involvement in the following Software Engineering activities in the last 10 years. (Case A: Q14, Case B: Q17) Options: {None, Some, Substantial} experience

The participants were provided with a full list (and descriptions) of the software engineering activities, as defined by Sommerville [Somm11a, fig.2.7], and were asked to rate their experience with them.

![Experience distribution](#)

Figure 6.27: Experience in the software engineering activities — distribution of responses.
As is apparent from Figure 6.27, the least represented expertise was software component construction (programming), while the most was stakeholder requirements elicitation. This is not surprising given that the representative sample’s inclusion criteria for Case A & B was RE experience. (This question was made optional in Case A’s survey due to time constraints on the practitioners. Hence, while the distributions sum to 100%, it actually represents 63% of Case A’s participants).

It is promising for the survey’s robustness to see full coverage of the software engineering activities, given that some of the questionnaire’s questions required through-life knowledge of the projects. Out of the total 12 software engineering activities, Case A’s median number of ‘Substantial experience’ responses was 5 (mean=$\bar{x}=4$, std. deviation=$\sigma=2.5$), while Case B’s median was 3 (mean=$\bar{x}=3$, std. deviation=$\sigma=1.5$). In other words, the consulting context (Case B) appears to have practitioners with a narrower range of areas of expertise. In particular, practitioners in Case B are less likely to have substantial experience in both ‘user requirements elicitation’ and ‘component construction (programming)’ than in Case A$^{93}$. While it seems plausible that this could be a causal factor for Case B’s lower software project benefit realisation rates (given that the developers were less likely to have elicited or analysed the non-software problem to be solved), no significant correlation or difference in sample medians were found, and so there is no evidence to support claims that ‘role fragmentation’ [Shar03] is harmful for software projects, —at least in their realisation of value.

Finally, the potential usefulness of this thesis’ framework is increased when there is a strong divide between those who elicit the problem, and those who perform development of the solution. Case B’s responses especially fall into this category, and to generalise, so do software projects involving many stakeholders with different backgrounds and interests (including developers, e.g., consultants).

6.1.7.3 What type of customer were your projects for? (Case B: Q2) Options: (Military, Civil, ~Equal)

Case B’s respondents had overwhelmingly ‘Mostly military’ project experience (75%), with the remainder responding with ‘Mostly civil’ (19%), and ‘About equally split’ (6%). Conversely, Case A’s respondents had ‘Mostly civil’ project experience. (The question was not asked in Case A’s questionnaire, since extra questions reduce survey completion rates, and it was not deemed to be a useful question: Case A’s organisation is divided into Civil and Defence, and the representative sample was drawn purely from the Civil organisation).

Since the questionnaire asks about the respondent’s last 10 years rather than their tenure at the organisation, the experience base undoubtedly includes some projects from previous employment. This is likely to be one reason why 25% of Case B’s respondents reported non-military experience, and is also why exactly 100% Civil experience cannot be assumed for Case A’s respondents.

Finally, we can infer basic characteristics about the cases’ contexts, given that the respondents’ experiences were dominated by one customer type, and hence the external validity of the case studies is improved [RuHö08]; Given that Case B’s respondents have mostly military project experience, most

---

92 The mean & std. dev. help describe the distributions, but are not statistically valid (ordinal, not ratio).
93 Kendall’s rank correlation (programming vs. req’s): Case A’s $\tau=0.14$, $p=0.7$; Case B’s $\tau=0.6$, $p=0.01$. This difference is supported by the researcher’s observations of the practitioners’ activities at both organisations.
projects will have complied with similar standards, quality guidelines, cultural norms, and so on, — as stipulated and governed by the UK Ministry of Defence (e.g., the AOF or MODAF [Mod12]). Conversely, despite that Case A has internal standards for RE, compliance is not mandatory, and each business unit appears to have adopted their own ways of working suited to their particular work.

**Point #75: Both cases have different customer types (Military/Civil), hence different RE practice**

Case B’s projects will have mostly followed the MODs AOF (similar to IEEE 29148 [Ieee11b])’s RE standards (having User, System, and Software Req. Docs.), while Case A’s standards are more varied (from Volere, to IEEE 830 [Ieee98], to ad hoc).

6.1.7.4 How many years of experience do you have in the development / engineering / production of software? (Case A: Q2, Case B: Q16)

In the combined response set, the majority of the responses (85%) had at least 10 years of software engineering experience (95% confidence interval for that population: [0.69, 1]),

94 This is positive for the robustness of the survey, given that a 10 year period of software engineering experience was considered necessary to have experienced a large enough number of projects mature (as earlier discussed in Section 6.1.3). Additionally, this finding indicates that the respondents would be less likely to make the kind of basic RE errors that less experienced practitioners would, as well as that there is scope for value in training the practitioners in this thesis’ framework, given that they are not ‘career hopping’ and hence that RE skills would be retained. Comparatively speaking, we cannot reject the null hypothesis that the distributions of experience in Case A & B are the same (as evaluated by a Wilcoxon rank sum test: W=111, p=0.25). In other words, Case A & B’s distributions of years’ experience are not significantly different, and so it is not likely that years of experience has strongly biased responses to the survey’s questions.

Figure 6.28 shows that the median respondent had 15 years of experience, and that the majority had more than 10 years of experience. Hence, the survey questions’ ‘in the last 10 years...’ restriction did cause omission of 5 or more years’ worth of their oldest projects. To establish whether or not this influenced the survey, the respondents were asked if their responses on the redundancy/inadequacy of requirements and, software project value realisation in general, would have been different had the survey not limited their experience to the last 10 years of their career (Case A: Q16.b, Case B: Q19.b). Promisingly, the vast majority (100% in Case A and 93% in Case B) of those who responded (64% in Case A and 88%), believed that there would have been ‘no difference’ (out of the response options {Better, Worse, No Different}), hence indicating that:

- The 10 year restriction did not change the conclusions of the survey, and;
- No significant reduction of the problem has been made despite advancements in the field.

94 While the same can be said for Case B’s population (94% of the responses, 95% CI: [0.72, 1]), it cannot for Case A’s population (73% of the responses, 95% CI: [0.44, 1]) primarily due to its sample size.
6.2 Summary of Questionnaire (and Interviews Expanding the Questions)

25 points of interest have been concluded (in the blue boxes) based on evaluation of responses to surveys, interviews with practitioners, and analysis of software engineering documents. The primary revelation is that implemented software capabilities are wasted as a result of inadequate RE. Furthermore, the participants believe that there is scope for the thesis’ framework to reduce those problems, and hence for the framework to be useful in practice. However, there are some areas of practice that would need to change to enable this thesis’ framework, namely that:

- Higher level desires of system owners were more often not available, than available, indicating that assessing the value of software requirements in terms of those desires may be difficult (Section 6.1.6.17).
- Requirements and objectives are not frequently quantitatively defined, which violates best-practice and the philosophies of the thesis’ framework (6.1.6.19), as argued in Chapter 2.

6.3 Feedback on the Usefulness/Usability of the Framework and Tool

Overall, the practitioners thought that the framework would be useful in communicating value propositions and exposing faulty assumptions, thereby reducing the risk of software project value failure. However, the general opinion was that the current state of practice in the organisations was not sufficiently geared toward RE to make the thesis’ framework reasonable in all of their software project contexts, but rather in the high-risk and high-value projects. In the lower-risk projects (i.e., where the likelihood or consequences of producing inadequate software is low), the practitioners believed that the a numerically ‘lighter’ version of the framework would be more cost effective.

Specific feedback from practitioners was gained from interviews and one-to-one sessions demonstrating the tool. This feedback is split into the two categories of perceived usefulness and limitations.

6.3.1 Perceived limitations (paraphrased from practitioners)

a) The approach mostly focuses on the benefits side of the ‘cost-benefit’ equation of a requirement. Although cost estimation tools such as COCOMO are referenced, it would be useful to build in better integration of software requirement cost (or effort) modelling.

b) The tool, while operational within the scope of this thesis’ requirements, is not a full scale RE tool. For example, version control of requirements, integration with Integrated Development Environments and Computer Aided Software Engineering tools is missing. So, although the tool could be used to implement the approach, some work would be required.
to make the tool interoperate with tools such as DOORS (for requirements repository management) or MagicDraw (for UML traceability).

c) The expected benefits of senior managers, or of strategic business units, are often ambiguous on purpose. The strategic direction (especially details such as numerical measures) is often secretive until the plan is made operational. Therefore, even though clear ‘Line of sight’ from a practitioner’s tasks to the organisation’s goals, has proven to be beneficial, this will not always be possible in every context. To resolve this, there might have to be different ‘views’ of the same model, where one view is a subset of the other view, such that it could be shared with practitioners without revealing secret information.

d) In large organisations, software projects are sometimes broken down into sub-projects, which are given to contractors to complete. Sometimes those contractors subcontract some software development work, and so on. Some of the interviewed practitioners have worked at the end of these chains, i.e., on projects far removed from the end-users or the stakeholders who would benefit. In such projects, the software development task is considered by the line managers as simply following the provided specification to the letter, and any attempts to go through the long communication chains to elicit value concerns would either not be practical (e.g., due to month long delays) or not possible (e.g., due to security or privacy reasons). In such cases, it is the ‘root’ organisation (that contains the end-users and/or sponsoring stakeholders) who should construct the value models.

e) The approach has the possibility for practitioners to ‘fudge the numbers’ in order to make the numbers support what they want to do (which might only be evidenced by qualitative knowledge). However, given that these ‘fudged’ numbers would be described in terms of objective scales of measure, rather than subjective scales, then such ‘fudging’ would be more easily refuted at peer review.

f) The tool’s ability to re-use models would, in an industry context, have to include access controls. In other words, it would be wrong to suggest requirements to the user, if the user does not have the security clearance of privilege to see those requirements. A role based access policy could be investigated.

g) The ‘dose-response’ functions between a software requirement and an expected benefit, are often only apparent after the software requirement has been implemented. In response to this, is that if the tool’s library of previous projects is built up, then the dose-response functions can be generalised by analogy, and hence re-used. Furthermore, the dose response functions, even if they are uncertain, are accurate representations of the practitioner’s state of mind at the time of making decisions. Therefore, uncertainty in the dose-response functions is not a limitation of the approach, but rather a positive property: it models the current understanding (and therefore exposes lack of knowledge as risks). One practitioner found it useful to think about modelling dose-response relationships under uncertainty, with the idea of using either a fine pencil to draw the line on the xy graph, or using a fat tipped board marker. I.e., where uncertainty (interval) is represented using the thickness of the line.

h) Re-use of the ‘dose-response’ cannot be ‘copy and pasted’ (without review) between similar functions between projects. Context is extremely important, especially the as-is values (which determine the domain of the dose-response functions). E.g., a 100ms reduction in service response time for google will have very different effects than for a small company’s web service, primarily because google’s response time is likely much faster than a small company’s.
### 6.3.2 Perceived usefulness (paraphrased from practitioners)

a) A common issue in software projects is dealing with stakeholders having different concerns and interests. Explaining software requirements in terms of benefits to be expected for each stakeholder’s department is a great way of getting more people interested in being involved in the requirements elicitation and prioritisation exercise.

b) Chapter 2’s conceptual framework was considered highly useful by practitioners. Without it, the resulting goal models would only convey pictures of ‘boxes and arrows’, and guidance to ‘draw another arrow leading to another box’ (adding another benefit to an existing benefit) would be severely lacking in guidance. If you asked a practitioner to ‘think of a benefit of x’, then it is unlikely to be defined with the clarity that framework achieves, and that is mostly due to its understanding of ‘benefit’, which is set out in Chapter 2.

c) In brown field development projects, where a project builds upon existing functionality, and therefore the requirements include existing functionality:

   Modelling requirements in terms of beneficial changes to be made to numerical measurements in the application domain avoids having to read huge lists of requirements that don’t explain what the project is good for.

d) The approach’s admission that sometimes a qualitative description will have to suffice instead of a quantitative description (of requirements and/or their causal propositions), is good. Industry puts time pressures on projects, and so the ability to follow a qualitative path through the approach is worthwhile.

e) Modelling precisely (conceptually) what is to be benefited by a software requirement helps to understand why that requirement should be implemented, and therefore gain a better understanding of the development task, including better ways to implement it.

f) Explaining the rationale for a implementing software capabilities helps to reduce the risk that management will, at a later date, question why the project’s direction took the direction it did. If the reasons are explained in clear models, and those models can be shared with (and understood by) management, then the responsibility is shared with them.

g) The tool’s ability to re-use models from previous projects is highly useful. Firstly, it improves Knowledge Management. The tool would act as an expert system, making use of huge libraries of knowledge that won’t leave the organisation. Secondly, it improves the motivation for making the requirements (and their benefits and/or obstacles) explicit; Rather than just being useful for one project, they could be useful for many projects.

h) The tool’s automatic drawing of the goal graph diagram was considered to be a necessary feature for implementing the framework, rather than a ‘nice to have’. The GRL diagrams became complicated quickly, e.g., with relationship lines frequently intersecting each other. The ability to filter (i.e., query) the model to draw only the elements of interest is key.
7. Conclusion

“Problems worthy of attack prove their worth by hitting back.”

Pit Hein
In this final chapter, the conclusions to the research problem and questions are discussed along with the main contributions and constraints. Scope for future work on the problem is outlined and indicators of research quality are evaluated.
### 7.1 Research Questions Revisited

The research questions that were first defined in Chapter 1, and then elaborated in Chapter 3, are revisited and answered in summary in Table 7.1.

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Summary of Answer (&amp; Reference to Full Answer)</th>
</tr>
</thead>
</table>
| **1** What is the motivation for predictively modelling the value of software requirements? | Chapters 1 (1.1.4) and 2 (2.2) described and evaluated claims from the literature that, roughly half of all implemented software requirements go on to be redundant, and that the majority of software projects do not realise their intended benefits.
Chapter 6 explained the research findings from this thesis' case studies that support such claims from the literature, and crucially explore the reasons for wasteful requirements (the top reasons included poor validation, communication, and solution orientation of requirements).
Finally, it is argued that explicit predictive models of value can help to reduce such failures by clarifying and communicating the assumptions and mechanisms that the value would depend on, so that inaccuracies can be resolved earlier on (and cheaper). |
| **2** What are the requirements for an approach to model the value of software requirements? | Chapters 4 describes the requirements for such an approach, elicited from the literature review of the important concepts (Chapter 2) and from interviews with practitioners. The main requirement from industry was to be able to expose and communicate the mechanism by which the requirement would cause value creation; The managers in industry previously had bad experiences with unrealistic stakeholder value expectations, and engineers developing solutions for no real business benefit. |
| **3** To what extent are current approaches capable of modelling the instrumental value of software requirements? | As evaluated in Chapter 4, the vast majority of approaches in the literature suitable to the problem are qualitative. Hence, they insufficiently model the causal mechanisms, and only serve to reinforce existing beliefs, rather than to challenge them.
Furthermore most do address uncertainty, which hinders a practitioner’s motivation to provide estimates, as well as comprehension of the likelihood of the value.
Finally, most do not have tool support (and so are difficult to apply in industry), and even fewer support the re-use of project data — which is a shame considering the high effort required to create the models and the little effort required to re-use them. |
| **4** Can a sufficiently capable framework for modelling the instrumental value of software requirements be designed? | A tool supported framework with the capabilities that were determined to be important was designed and exemplified in Chapter 5. Considerable effort was spent optimising the framework’s techniques for their pragmatism rather than for rigour on principle. Academic and industrial desires for what makes a framework ‘good’ were balanced, given that the researcher both was embedded within industry, and had papers on the framework accepted at academic peer review.
The framework was developed with feedback from industry, where the framework and tool were regularly exposed to practitioners. The majority of practitioners reported that they would use the framework & tool, but expressed concerns over integration with current business processes and standards, as well as the time available for RE. |
| **5** What are the lessons learnt concerning the implementation of the framework? | The lessons learned from implementing the framework in industry are discussed in Chapter 6. In summary, practitioners (still) do not afford RE much effort (as compared to the 2-4% of project effort typically spent [Fire04]). Hence, trying to introduce a quantitative (i.e., more ‘heavyweight’ than qualitative) framework for RE received little enthusiasm (mostly due to the ‘What’s in it for me?’ problem). Thus, the framework’s qualities that improve usability should be advertised to practitioners (its qualities that improve utility are of more interest to managers). Specifically, the framework and tool’s ability to automate the visualisation of means-ends graphs, that explain and risk-reduce the design rationale for requirements, proved to be a good ‘selling point’.
Overall, it is recognised that the state of practice in RE is lagging far behind the state of the art, and hence the framework’s more advanced features (i.e., those requiring more input from stakeholders) are less likely to be adopted in the current ‘RE climate’. |
Overall, the overarching problem statement defined in Chapter 1 was to establish whether value realisation problems caused by RE exist in industry, and then to understand whether a systematic tool-supported approach could be designed to be effective in reducing or avoiding those problems. Given that the problem was investigated (Chapters 2 and 6), a tool supported approach was designed and implemented (Chapters 4 & 5), and industry practitioners perceived it to be useful (Chapter 6 & 7), the thesis is considered complete.

7.2 Future Work

As a result of research on answering this thesis’ main research questions, a number of future research questions were identified (often requiring more resources than this PhD project had):

- Can a library of patterns (as in Jackson’s Problem Frames [Jack00]) be established for various application domain contexts to describe generic software capabilities suitable for realising stakeholder value, (especially for Zuboff’s ‘Inform’ & ‘Automate’ types of IT, — given that the ‘Transform’ type of IT is inherently dynamic and ever-changing [Zubo85])?
- Can a library of benefits resulting from software requirements be established that are considered to be sufficiently intrinsically valuable (with annotation of associated risks), (a low-risk example is ‘increasing profit’), such that modelling effort can be stopped once they are reached in a chain of instrumental value derivation?
- Can estimators who estimate the value contribution a software requirement be calibrated, such that their accuracy and confidence in their future estimates is more credible?
- Is redundancy of software features correlated with perceived success of software projects in a wider context than this thesis deals with, or is it by now considered to be part of the essence of software engineering (determining the effort spent by practitioners to reduce it)?
- Investigate the usefulness of Monte Carlo simulation (as guided in [Cant09, Thom12]) to reduce uncertainty in the predicted value contributions of software requirements.
- Investigate the usefulness of modelling temporality as an input to utility functions and descriptions of an expected benefit’s measurement scale (on the premise that ‘what hurts now may kill in the future’).
- Investigate the best technique (usability and utility) for traceability down from value expectations to the software design and components that operationalise it (rather than just requirements), e.g., with SysML and associated Computer Aided Software Engineering tools.

To answer these research questions, it would be necessary to evaluate them within an organisation over a long time period. Hence, it would be highly beneficial if the researcher (or a research project team member) primarily took the role of the industry practitioner who implements the framework, given that they cannot be answered without strong adoption and frequent usage of the framework.

As a result of the research on analysing the extent to which current RE practice does not adequately explain and trace requirements to their problems (and hence their benefits)\(^95\), a number of future research questions were identified, (as were listed in the paper [ELDK14]).

---

\(^95\) To use Systems Engineering terminology: the extent to which the User (Stakeholder) Requirements documents and System (including Software) Requirements documents did not fulfil their roles. As part of this thesis’ research, Natural Language Processing was used to explore these specifications’ quality [ELDK14].
• Is there a correlation between software project success (or either the amount of re-work or ad-hoc communication required to make it successful) and the compliance of User Requirements Documents and System Requirements Documents to requirements engineering standards, in the defence IT context and (perhaps later) the wider IT context?

• Can a greater number of trivial differences be identified in UR-SysR pairs by extracting and comparing semantic information, such as by using domain ontologies and lexical databases (e.g., Princeton’s Wordnet), or by comparing ‘Parts of Speech’ (as are extracted from requirements in [BSBG13])? This would elucidate whether RE authors purposely replace SysR terms with synonymous terms to form a UR that appears to be different.

• Can machine learning techniques (e.g., a Naïve Bayesian classifier) be used to better classify UR-SysR pairs as to whether useful information is added by the link or not?

7.3 Contribution

To the best of this researcher’s knowledge (and the opinions of the peer reviewers of the published papers), prior to this thesis, an approach did not exist in the literature that enabled the effect of a software requirement’s satisfaction on its intended benefits to be modelled and analysed [Elli13]:

a) at varying levels of goal satisfaction extent (explaining the effects of partial/full requirement satisfaction on multiple levels of goal abstraction);

b) at varying levels of stakeholder confidence (explaining the extent to which a requirement’s satisfaction may not contribute to a goal as specified);

c) at varying levels of software usage (explaining the different profiles of software usage that could affect a requirement’s contribution to higher level goals);

d) at varying levels of stakeholder utility (explaining the non-linear relationships between the extent of a goal’s satisfaction and the stakeholder satisfaction to be gained);

e) at varying levels of stakeholder agreement (explaining the variance between the stakeholders’ estimates about the benefits that will be contributed);

f) with limited prior knowledge about the business goals (assisting the elicitation of the intrinsic values that the software features are intended to be instrumental for realising);

g) with tool support to describe, simulate, visualise, validate, communicate, re-use, recommend, and assist the modelling of a software requirement’s value creation mechanism(s).

Furthermore, reasons for and consequences of wasteful software requirements had not been studied in detail or in particular application contexts (as they are in Chapter 6), prior to this thesis. For example, poor RE (especially Validation), was considered to be a top cause of wasteful implementations. 28 fundamental concepts were defined and related in Chapter 2, providing a theoretical foundation for understanding this thesis’ problem, spanning many fields of research and practice. 27 existing approaches that could be applied to this thesis’ problem were compared in Chapter 4, providing a unifying comparison framework for quickly understanding the previous work. Additionally, the degree to which current RE practice adequately describes both problems and solutions had not been studied, as is primarily treated by one of this thesis’ papers [ELDK14].
Finally, this thesis has resulted in a number of academic papers in conferences and journals:


Two additional papers are in progress resulting from this thesis’ work. The first concerns the survey data on the incidence and causes of wasted requirements implementations, and the second concerns the framework’s tool’s use of machine learning techniques (recommendation) to re-use goal models from previous projects.

### 7.4 Research Quality Checklists

Kitchenham et al.’s checklist for the quality and applicability of research in software engineering for both research and practitioners [KABB06] is well accepted in the RE community [DSCH11]. (Given their generality [KABB06, p.47], they are similar to Runeson & Höst’s guidelines for case study research in software engineering, which were also considered [RuHö08]). Table 7.2 demonstrates compliance with Kitchenham et al.’s guidelines from the perspective of other researchers, while Table 7.3 does so with their guidelines from the perspective of practitioners, (minus those that were already covered by the answers in Table 7.2).

#### Table 7.2: Kitchenham et al.’s Research Checklist from other researchers’ perspective [KABB06]

<table>
<thead>
<tr>
<th>Checklist Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Is the research problem hypothesis easy to identify?</td>
<td>The research aims and objectives were defined in the form of hierarchical research questions (as is suggested practice in Design Science research [Wier09a]) in Chapter 3 (Research Method). Falsifiable hypothesis were derived from these questions and evaluated in Chapter 6 (Case Studies).</td>
</tr>
<tr>
<td>2. Is there an underlying causal model? If so, what is it?</td>
<td>Causal factors leading to this thesis’ problem were explored and discussed in Chapter 2 (2.2.2), and the effects were discussed in Chapter 1 (1.1.3).</td>
</tr>
<tr>
<td>3. Is the terminology defined and explained?</td>
<td>Chapter 2 (2.1) introduces, defines, and relates the concepts and constructs that this thesis’ framework builds upon, and whose research questions explore. Literature is cited to academic standards and evaluated in the context of this thesis’ problem.</td>
</tr>
<tr>
<td>4. Is the research related to other relevant research?</td>
<td>Chapter 2 explores the wider relations of the thesis’ requirements engineering research problem to fields as wide as philosophy and epidemiology. Chapter 4 evaluates and compares the relevant requirements engineering research with this thesis’ framework (Section 4.9).</td>
</tr>
<tr>
<td>5. Is the experimental design appropriate?</td>
<td>Chapter 3 (Research Method) evaluates and justifies the research method, design, and tools used to answer the research questions.</td>
</tr>
<tr>
<td>6. Is the statistical analysis correct?</td>
<td>The rationale and methods for the statistical analysis of the data used to answer the research questions is described in Chapter 6 (6.1.5).</td>
</tr>
<tr>
<td>7. Is the raw data available?</td>
<td>The raw data from the questionnaires, interviews, and the source code for the framework’s supporting tool, is made available on a CD ROM. (Hence copies of the data will be available to access via the universities’ library).</td>
</tr>
</tbody>
</table>
8 Is it easy to identify the findings / results of the experiment(s)?

Do the conclusions arise from the results?

Is the argumentation clear?

Are limitations of the experiment made clear?

Is there any discussion of required further research?

Summaries of important findings and results are included throughout the thesis in blue boxes, so that the important points can be quickly found. They are placed immediately after the discussions of those findings and results, so that it is clear how those conclusions were reached.

Chapter 6 includes a discussion of the limitations and threats to validity of the inferences and assumptions made to answer the research questions.

Interesting research questions following on from this thesis are proposed in this Chapter (7).

Table 7.3: Kitchenham et al.’s Research Checklist from the practitioner’s perspective [KABB06]

<table>
<thead>
<tr>
<th>Checklist Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Is the research relevant?</td>
<td>The context motivating this thesis was defined in Chapter 1 (1.1.3), and the context of the problem and intended use for the proposed framework was defined in Chapter 2 (2.3).</td>
</tr>
<tr>
<td>Is the application type specified?</td>
<td></td>
</tr>
<tr>
<td>In what context is the result/claim useful/relevant?</td>
<td>The proposed framework benefits from software tool support to reduce the complexity of its application, as well as to enable several of its key capabilities. The tool is written in Java using the Eclipse Framework, is configured to use an embedded database (SQLite), and is bundled with the Java Runtime Software. Hence, it does not depend on the existence of a particular operating system, database infrastructure, or software. The tool is open source and available to download online.</td>
</tr>
<tr>
<td>3 Is the availability of the required support environment clear?</td>
<td></td>
</tr>
<tr>
<td>4 Are any technology pre-requisites specified?</td>
<td>Personal qualities and capabilities required for the facilitation of the proposed framework are defined in Chapter 2 (2.3.3). Walkthroughs of the tool’s use are included with the download of the compiled software (and in the Appendix of this thesis).</td>
</tr>
<tr>
<td>5 Are the experience or training costs required by development staff defined?</td>
<td></td>
</tr>
<tr>
<td>6 Do the results apply and scale to real life?</td>
<td>The following demonstrates that the research applies to real life and has practical relevance:</td>
</tr>
<tr>
<td>7 Is the experiment based on concrete examples of use/application or only theoretical models?</td>
<td>• This thesis’ problem was identified from the researcher’s time spent with its industrial partners, rather than from academic origins.</td>
</tr>
<tr>
<td>8 Is new approach / technique / technology well described?</td>
<td>• The research questions were answered using data elicited from industry practitioners and analysis of software project documents.</td>
</tr>
<tr>
<td>9 Are the conclusions/results useful (&amp; shown how they can be used in practice)?</td>
<td>• The framework was developed with iterations of feedback from industry, was exemplified using real industry software projects, and was evaluated with the opinions of experts within industry.</td>
</tr>
<tr>
<td>10 Is the expense involved in adopting the approach defined (i.e., Return On Investment)?</td>
<td>Estimates of the time taken to apply the framework (and related software tool capabilities) are provided in Chapter 6, as derived from real industry usage. Evaluating the ROI of a value based requirements engineering method takes more time than was available, in order to understand the usage of implemented software requirements, as well as reductions in scrap, re-work, and inadequacy of the resulting software capabilities throughout its life.</td>
</tr>
<tr>
<td>11 Are Technology Transfer issues discussed?</td>
<td>Risks and limitations, as well as motivators and de-motivators for the framework’s adoption are provided in Chapter 6.</td>
</tr>
<tr>
<td>12 Are any risks associated with adoption defined?</td>
<td>It is promising that none of the questions in either checklist (Table 7.2 &amp; Table 7.3) were left unanswered, but ultimately the degree to which this thesis complies with the checklist items is the reader’s personal judgement. Given fewer time and resource constraints, the extent of the thesis’ compliance would ideally be evaluated by asking expert researcher and practitioners to rate the checklist questions on a Likert scale of agreement, and then by assessing the inter-rater reliability [Gwet14].</td>
</tr>
</tbody>
</table>

7.5 Research Constraints

The research was motivated by, and investigated for two particular companies (in an effort to increase pertinence of the research). Consequently this inherently limits the applicability of its
conclusions (and to some extent the applicability of the resulting artefacts) to the context of those two organisations (IT defence consulting, and aeronautical engineering). However, care was taken to generalise, as well as to limit external validity in conclusions (in Chapter 6). Furthermore, the contexts for which this thesis’ primary artefact (the tool supported framework) is applicable have been thoroughly described (in Chapter 2.3). Crucially, they are not limited to the contexts of particular industries, but rather to characteristics of an organisation and a project.

The largest constraint on this thesis was time and resources. The researcher spent 2 years at two different companies, participating in the day-to-day lives of software and requirements engineers. This has obvious implications for the time available to perform research tasks. Had more time been available, a more comprehensive evaluation of the framework would have been performed. Primarily, the hypothesis that the following activities (as defined in Chapter 2.3.1) could be aided by the framework would have been tested using more rigorous methods than expert opinion:

A. Elicitation of new Candidate Requirements
B. Approval of a Candidate Requirement
C. Specification of an Approved Requirement
D. Release Planning of a Specified Requirement
E. Verification & Validation of an Implemented Requirement

Furthermore, it took more than one year to process the legal contracts between the researcher’s institution and the industrial organisations, which significantly limited interaction with industry practitioners and their resources (documentation, IT facilities, etc.). This meant that rather than the ideal ‘problem exploration first, solution design second’ sequence, the researcher spent more time investigating solutions and programming the tool in the first phase of the project than would have been ideal (since it was not yet possible to perform primary research on the problem). This ironically, caused some of the software tool’s capabilities to be ‘Scrapped & Re-worked’ due to an immature understanding of the problem that the industry partners desired to be solved.

Finally, given that the researcher spent considerable time embedded within two industrial organisations, ‘expert opinion’ evaluation (and the constant feedback gained from frequent exposure of the tool to practitioners) is subject to social and political biases. However, the researcher was careful to remind the practitioners that the researcher was motivated by successful research, rather than producing a successful artefact (approach or tool). In other words, they were told that it did not matter if they thought a developed capability was not useful.

7.6 Lessons Learned from the Research Project

The primary lesson learned from the research project is that it is difficult to evaluate solutions for problems that practitioners are not highly personally motivated to solve (as is often so in organisations). Hence, management buy-in is not just nice to have, but is essential (as demonstrated by the difference in survey response rates between Case A & B, where B had ‘Director’ sponsorship).

For example, project teams could use both this thesis’ framework, and the closest competing approach identified in Chapter 4’s review, which would be either GRL + (Key Performance) Indicators [HBJY12] or B-SCP (OMG’s BMM + i*) [BCVP06]. Initially, the practitioners could rate and compare the ease of use and the usefulness of this thesis’ approach, but eventually, objective measures should be used to decide on the usefulness and usability of the technique. Such measures include the amount of implemented requirements scrapped or re-worked, the frequency of developer→user clarification of requirements, the perceived ‘line of sight’, and so on (see Chapter 1’s description of the problem).
This researcher now has a strong appreciation for the difficulty of embedded academic research into industry, and especially acknowledges the necessity of selling practical ‘what’s in it for you?’ benefits to business stakeholders. Unfortunately, many aspects of academic research in requirements engineering have little direct benefit to industry practitioners [BDFG05], compared with what the practitioners could have alternatively achieved using the time spent with the researcher, and incentives are often limited by ethical or organisational standards. Hence, in future research projects in industry, focus should be placed on easing the practitioner’s jobs\(^97\) (i.e., improving some problematic aspect in software engineering that affects the usability of something) if high researcher-practitioner participation is valued.

### 7.7 Recommendation for Future (Implications on Practice)

The key ‘takeaway point’ from this framework is that whenever a system capability is planned, the requirements engineer should think about the various causal levels of an uncertain quantity to be influenced in an application domain that would actually entail the realisation of stakeholder value.

Real evidence of this thesis’ problem, and an exploration of the causes from particular industry contexts now exists thanks to Chapter 6. This should be used to motivate practitioners to take more of an interest in the value proposition of their software, rather than just the software.

Requirements Engineering, as embodied by the principles of this thesis, is on some occasions like the wheel in Figure 7.1. The up-front investment of time and resources spent on analysing the problem (equivalent to developing and fitting the wheel) has the capacity to improve the project through reduced re-work or value failures caused by misunderstandings. There remains considerable work to motivate that investment in practice.

---

\(^{97}\) Rather than on improving something belonging to someone who the researcher is not heavily interacting with, e.g., managers or the stakeholders who expect benefit from the software.
8. References


References


Data for All Employees


[Busi14b] BUSINESSDICTIONARY: What is a Representative Sample? definition and meaning. URL http://www.businessdictionary.com. - retrieved 2014-03-06. — What is a Representative Sample? — Representative Sample: A small quantity of something such as customers, data, people, products, or materials, whose characteristics represent (as accurately as possible) the entire batch, lot, population, or universe.


[Cice45] CICERO: de Finibus Bonorum et Malorum (The Extremes of Good and Evil). 45AD


References


LuP99


MaKh14


References


[Mini05] ‘MINISTRY OF DEFENCE’: MODAF Community of Interest Deskbok [Nr. MODAF-M10-004] : MoD, UK, 2005. — v0.4


References


[Wier05] Wieringa, R.J.: Requirements researchers: are we really doing research? In: *Requirements Engineering* vol. 10 (2005), Nr. 4, pp. 304–306


9. Appendices

This page is intentionally blank.
Appendix A - ‘GoalViz’ Tool Screenshots

This document is intended to complement Chapter 5 of the thesis, by showing the tool created to support the framework. The steps, as well as the reason for following them, are explained in Chapter 5 in great detail.

Step 1: Open the tool, create a new file

The file created here will be used to re-open the project in the future. All changes are automatically saved to the file.

Step 2: Create some software requirements

Step 2.1: Open the tool and create a new folder
The tool is split into three vertical (resizable) panes. The left pane provides a tree view of the model elements, e.g., the software requirements or the expected benefits. (The tree view has been programmed to support the directed acyclic graph data structure, as opposed to the tree data structure, hence one element may have multiple parent elements.) The middle pane provides the automatic visualisation of those elements using the GRL graphical notation. The right pane shows the properties of the selected element, the links to other elements, and the textually similar elements. Data binding of the GUI elements to the model ensures that all views of the model are kept synchronised.

**Step 2.2: Add lower level requirements by decomposing the high level requirement**

By clicking on the ‘new requirement’ button (or by pressing ‘enter’ on the keyboard while a requirement is selected in the tree view), a new requirement is created, and is linked to the selected requirement with a GRL decomposition link (default). The visualisation in the middle pane is automatically generated when the tree view’s selection changes.
Step 2.3: Complete the decomposition of requirements

Step 2.4: (Optional): Change visualisation options

By clicking on the buttons at the bottom of the middle pane, the visualisation can be instantly changed.

Here, the visualisation is forced to consider the grouping, as well as the decomposition level of the model elements. This is useful where the drawing algorithm’s placement of elements is not true to the model (especially the ‘ranks’).
The model visualisation is interactive, hence model elements can be selected from the visualisation, zoomed, panned, and so on. Here, the visualisation is changed to lay out linked elements horizontally rather than vertically.

**Step 2.5: Populate the attributes of the requirements such that their satisfaction can be tested**
Step 3: Add expected benefits

Step 3.1: Create a new folder that will contain the model elements

Click the ‘add folder’ button in the bottom-left pane. Then, name the folder as the actor that will own the benefits. Move the folder up a rank to be above the ‘Software Requirements’ folder, by clicking on the ‘up’ arrow.

Step 3.2: Add a new expected benefit to that folder
Populate the fields for the expected benefit, making use of the text fields’ auto-complete values and their tooltips.

**Step 3.3: Quantify the targets of the expected benefit (i.e., how much benefit is expected?)**

Using the slider GUI element, specify the numerical targets, in the context of the ‘scale’ and ‘units’ textboxes. By changing the dropdown box ‘Higher is better’ to ‘Higher is worse’, the colours in the slider GUI element (red-green) swap position. This allows the tool to understand whether bigger numbers on the scale are better or worse. The ‘precision’ textbox can be used to distinguish between continuous or discrete variables. Specifically, it defines the smallest amount of change on the scale, i.e., the amount added or subtracted by moving the slider by just one notch. Finally, specify the timeframe and the scope of the measurement to be made of the expected benefit’s scale.
Step 4: Link the software requirement to the expected benefit

Step 4.1: Create the contribution link (drag and drop)

In the tree view pane, drag the software requirement to the expected benefit that it is predicted to contribute to. A window will appear to request the link direction and type. Before this window appears, the link will be validated against the current model, in order to prevent redundant links (e.g., in an existing chain of A→B→C, creating a link between A→C), or cyclical links (e.g., creating A→B→C→A).

As soon as the link is created, the tool’s GUI elements update to reflect the new model, e.g., the green highlighting in the tree view, the automatic drawing of the goal graph, and the list of the select element’s links in the right pane.
Step 4.2: (Optional) Change visualisation options

In the above screenshot, the expected benefit is drawn on the same rank (i.e., vertical position) as the software requirements. This does not make for easy comprehension. The tool’s ‘Rank’ function was created to address this, by modifying the automatic drawing algorithm to take into account an element’s level of decomposition and folder.

There is still room for improvement in the automatically drawn goal graph. If the leaf level requirements are linked to many expected benefits, then the diagram will become messy, since the links will have to curve around the requirement’s parent requirements. The figure below provides an example of a link that has to curve excessively.

The ‘reverse requirements’ function was created to address this problem, by reversing the direction of the requirement’s links in the diagram, as shown below:
Finally, in order to concentrate on one chain of requirement→benefit contribution, it is possible to disable the drawing of the selected element’s parent’s branches, (the ‘All Links’ button). Compare the above and below screenshots.

**Step 5: Estimate the size of the contribution**

**Step 5.1: Select the contribution link in the goal graph visualisation**

In the goal graph drawing, click on the contribution link between the software requirement and the expected benefit.
Step 5.2: Specify the table function for the estimated contribution

In the bottom right corner, using the editable table cells of the auto-adding table rows, specify the table function for the contribution. In the above example, the table function reads “If the software requirement’s implementation status is true, then there would be a 6 hours reduction in the average time spent per month to remove expired adverts in the venue”. In the 2nd (rightmost) screenshot, a blue triangle shows the effect of that table row’s reduction on the scale.
Step 6: Explore the layers of instrumental value

Step 6.1: Add more expected benefits for the respective actors (above the existing expected benefit)

The two expected benefits at the top of the chain are shown here to be empty, since they have just been added to the tool. The example goal graph used in this document mirrors the example used in Chapter 5 of the thesis, and so if the reader is curious about what those expected benefits might be, they should refer to Chapter 5’s figures.

Step 7: Explore ‘what if?’ questions (e.g., what if the requirement is/not implemented?)

Step 7.1: Click on the ‘Visualise link to root’ button in order to see the contribution’s propagation

After clicking on the Visualise Link to Root’ button at the bottom of the middle pane, a blue line is overlaid on the goal graph, as shown in the screenshot on the right. This line represents the contribution made by the leaf (bottom)
element, propagated up toward the root (top) element. The semantics of the blue line’s x-axis position are follows:
On a goal graph element, the leftmost x-axis position represents zero satisfaction of that element’s target level of satisfaction. The rightmost x-axis position represents full satisfaction. X-axis positions in between the leftmost and rightmost positions represent the (percentage) degree of satisfaction. In summary, the blue line represents the effects of the contributions of a chain’s elements on those element’s target levels of satisfaction.

For example, the leaf element’s (software requirement’s) blue line is overlaid as far right as is possible, indicating that in this modelling scenario, it would be fully satisfied (i.e., ‘implemented?’ is true rather than false). From the blue line’s position on the next model element up (which is highlighted in the blue colour, to indicate that it is the currently selected item in the tool), we can see that it is fully satisfied as a result of the child element’s contribution, since its blue line is at the rightmost position. To understand the reason why, examine the specified contribution table function (‘if-then-confidence’) and the slider GUI element (‘hard goal attributes’) in the left screenshot, which states that:

- the contribution of ‘6’ [hours reduction] if the requirement’s implementation status is ‘true’;
- the ‘as-is’ value of the expected benefit of ‘7 hours’, and its target of ‘1 hour’;

Hence, if a ‘6’ hour reduction from a ‘7 hours’ as-is value is contributed, then the target is met. If the modelling scenario changes to ask ‘what if?’ only a ‘5’ hour reduction would be contributed, then the blue line would indicate only partial satisfaction of the expected benefit, rather than full satisfaction. The next screenshot exemplifies this.
Step 8: Re-use models from previous projects

In the context of a new software project, a new model element is created (an expected benefit: to reduce the time taken to remove expired news articles) that is similar in theme to the previous project’s model element(s). To re-use the knowledge embedded within the previous project’s models, recommendations are provided to the tool’s user, as shown in the bottom-right of the above screenshot. As described in Chapter 5 of the thesis, the tool finds model elements similar to the currently selected model element, and then recommends that their linked items might be useful in the current model context. For example, perhaps the functional software requirement to ‘automatically delete media items via schedule’, might be applicable in the context of a news website, rather than of a digital signage media library (the original context). This could help to elicit software requirements, expected benefits, obstacles, actors, assumptions, and more.
Towards an Approach for Analysing the Strategic Alignment of Software Requirements using Quantified Goal Graphs

Richard Ellis-Braithwaite\(^1\)  Russell Lock\(^1\)  Ray Dawson\(^1\)  Badr Haque\(^2\)
\(^1\)Computer Science, Loughborough University  Leicestershire, United Kingdom
(r.d.j.ellis-braithwaite@lboro.ac.uk, r.lock@lboro.ac.uk, r.j.dawson@lboro.ac.uk, badr.haque@rolls-royce.com)
\(^2\)Rolls-Royce Plc.  Derby, United Kingdom

Abstract—Analysing the strategic alignment of software requirements primarily provides assurance to stakeholders that the software-to-be will add value to the organisation. Additionally, such analysis can improve a requirement by disambiguating its purpose and value, thereby supporting validation and value-oriented decisions in requirements engineering processes, such as prioritisation, release planning, and trade-off analysis. We review current approaches that enable such an analysis. We focus on Goal Oriented Requirements Engineering methodologies, since goal graphs are well suited for relating software goals to business goals. However, we argue that unless the extent of goal-goal contribution is quantified with verifiable metrics, goal graphs are not sufficient for demonstrating the strategic alignment of software requirements. Since the concept of goal contribution is predictive, what results is a forecast of the benefits of implementing software requirements. Thus, we explore how the description of the contribution relationship can be enriched with concepts such as uncertainty and confidence, non-linear causation, and utility. We introduce the approach using an example software project from Rolls-Royce.

Keywords—Requirements Engineering; Strategic Alignment; Quantified Goal Graphs; Requirements Traceability

I. INTRODUCTION

This paper describes in more detail the concepts and the technique originally presented at the 7\(^{th}\) International Conference on Software Engineering Advances [1]. It extends the work namely through a more comprehensive literature review, and the introduction of multi-point goal contribution.

The growth of the strategic importance of IT [2] necessitates the need to ensure that software to be developed or procured is aligned with the strategic business objectives of the organisation it will support [3]. Attaining this alignment is a non-trivial problem; firstly, decisions in the Requirements Engineering (RE) phase are some of the most complex in the software development or procurement lifecycle [4], and secondly, there is a gulf of understanding between business strategists and IS/IT engineers [5]. If alignment were trivial and easy, then it would not have been the "top ranking concern" of business executives for the last two decades [6], over 150 papers would not have been published on the topic [7], and perhaps there would be less software features implemented but never used (currently half of all features [8]).

Decisions made in the requirements phase greatly affect the value of the resulting software, e.g., in requirements prioritisation, the selection of the least important requirements allows costs to be cut by trading off the development of those requirements. The correctness of any such decision depends entirely on the availability of information about the choices available to the decision maker [9]. In this context, information about the value of a requirement, in particular, the causes and dependencies of value creation, is highly useful. Goal graphs are of great interest because they are well suited for visualising cause-effect, dependency, and hierarchical relationships between requirements [10].

This paper explores the suitability of goal graphs for demonstrating a software requirement’s strategic alignment. Current Goal Oriented Requirements Engineering (GORE) approaches primarily take a qualitative or subjective approach to describing goal contribution, such as GRL’s \(\{-,+,++,+\}\) or \([-100,100]\) scores [11]. As a result, any strategic alignment proposed by the use of goal graphs is not specific, measurable, or testable. Proposed extensions by Van Lamsweerde [12] do not consider that a chain of linked goals may contain a variety of metrics that need to be translated in order to demonstrate strategic alignment. Additionally, the certainty, confidence, and credibility of the predicted contribution are not explored. A probabilistic layer for goal graphs is proposed in [13], which recognises that goals are often only partially satisfied by software requirements. However, the (often non-linear [14]) effects of the incomplete goal satisfaction on an organisation’s various levels of business strategy are not explored. Furthermore, current methods do not consider how goal contribution scores are elicited [15], and how their calculation affects the credibility and accuracy of the claimed benefits. This paper attempts to demonstrate how the above problems can be addressed, thereby improving the applicability of goal graphs for the problem of analysing the strategic alignment of software requirements. By making assumptions about business value explicit, our approach complements Value Based Software Engineering (VBSE) [16].

We have developed and implemented our approach in partnership with an industrial partner (Rolls-Royce) to ensure its usability and utility in real world settings. We use examples in the context of a software project for a Business Unit (BU) responsible for part of a Gas Turbine (GT) engine, henceforth referred to as GT-BU. The software will automate geometry design and analysis for engine components, as well as for their manufacturing tools such as casting molds. Simply put, engineers will input the desired design parameters and the software will output the component’s geometry. At the time of our involvement with the project, some high-level
software requirements had already been elicited and defined according to the Volere template [17].

In Section II, we describe the problem that this paper addresses. Then, in Section III, we define and describe the essential terminology and concepts, while in Section IV, we present and evaluate the extent to which existing solutions address it. Section V presents our approach and tool in order to address the gaps outlined in Section IV. We conclude in Section VI with the paper’s contributions and future work.

II. THE PROBLEM

Stakeholders responsible for a software project’s funding need to be able to demonstrate that the software they want to develop or purchase will be beneficial. Decision makers require granularity at the requirement level, rather than the project level, since individual software functions or qualities may significantly affect the benefit or cost of the software’s development or procurement. Furthermore, stakeholders performing requirements engineering processes where the benefit of a requirement is questioned (e.g., in prioritisation, release planning, trade-off, etc.) need to know how benefit is defined by the stakeholders, and then how the requirements (and their alternatives) contribute to it.

As an example of the problem that this paper examines, we introduce the following high-level software requirement taken from our example project: “While operating in an analysis solution domain and when demanded, the system shall run analysis models”. The rationale attached to this requirement is “So that structural integrity analysis models can be solved as part of an automated process”. Although the rationale hints at automation, the requirement’s benefit to the business and the potential for alignment with business strategy are unclear. In order to understand the latter (i.e., the alignment with business strategy), the extent to which the organisation wants to reduce the problems related to manual structural integrity analysis needs to be understood (i.e., its business objectives). In order to understand the former (i.e., the business value), the extent of the requirement’s contribution to the problem to be solved needs to be made clear, e.g., the extent that automation is likely to solve the problems related to manual structural integrity analysis. For example, if the manual process costs the business in terms of employee time and computing time, how much time is consumed, and at what cost? Then finally, to what extent will the software requirement’s successful (or partially successful) implementation reduce the time consumption and cost?

To paraphrase Jackson & Zave [18], for every stated benefit (or an answer to “why?”), there is always a discoverable super benefit (i.e., benefit that arises from that benefit). For example, the slow and human resource intensive process may constrain the designer’s ability to innovate (by not being able to analyse new design ideas), which may ultimately harm the organisation’s competitive advantage. Many levels of benefit follow a requirement’s implementation. Each level provides the possibility of contributing to a business objective at a different level of the organisation. There are arguably more levels of benefit than it would be sensible to express within a requirement, since several requirements may achieve the same benefit, but their contribution will vary.

III. BACKGROUND TERMINOLOGY

A. Software Requirements

In 1977, Ross and Schuman stated that software requirements “must say why a system is needed, based on current or foreseen conditions” as well as “what system features will serve and satisfy this context” [19]. Robertson & Robertson later expanded the concept of a “feature” by defining a requirement as: “something that a product must do or a quality that the product must have” [17], otherwise known as functional and non-functional requirements, respectively. It is worth noting here that, according to the “what, not how” [20] paradigm, software requirements are often incorrectly specified in practice (i.e., they often describe how features should work, rather than what features should be implemented). Consequently, implementation bias may occur, unnecessarily constraining the design space. Practitioners are not entirely to blame however, since the what and how separation is confusing. This is because a requirement describes both concepts at the same time, but at different levels of abstraction. For example, “print a report” is what the system should do, but also how the system should “make the report portable” – which again, is what the system should do, but also how the system should “make reports shareable”. The how and why aspects of a what statement are simply shifts in the level of the statement’s abstraction (down, and up, respectively).

Popular requirements engineering templates (e.g., Volere [17] and IEEE Std. 830-1998 [21] and meta models (e.g., SysML [22] and the Core Metamodel [23]) tend not to focus on the why aspect, typically addressing it by stipulating that rationale be attached to a requirement. However, rationale is not always an adequate description of why the requirement is valuable. If only one why question is asked about the requirement then the rationale can still be distant from the true problem to be solved (i.e., the essence of the requirement), and it may be defined without consideration of its wider implications. A stakeholder’s “line of sight” (i.e., the ability to relate low-level requirements to high-level business goals), and thus, the ability to determine the value of a requirement, depends on their ability to find answers to enough recursive why questions. Anecdotally, empirical studies at Toyota determined that the typical number of why questions required to reach the root cause of a problem is five (thus spawning the “five whys” method popularised by Six Sigma) [24].

B. Strategic Alignment

Singh and Woo define business-IT strategic alignment as “the synergy between strategic business goals and IT goals” [7]. In the IT context, Van Lansweerde defines the term “goal” as a prescriptive, optative statement (i.e., desired future state) about an objective that the system hopes to achieve [25]. In the business context, a goal is defined as an abstract indication of “what must be satisfied on a continuing basis to effectively attain the vision of the business” [26]. In order to relate the goals of the system to those of the business, an integrated definition of the terms used by business strategists and software developers is required. Furthermore, the first definition does not make “objective” distinct from “goal”. The Object Management Group (OMG) defines such
terms in its Business Motivation Model (BMM) [26]. There, an objective is defined as a specific "statement of an attainable, time-targeted, and measurable target that the enterprise seeks to meet in order to achieve its goals". According to the definitions of goals and objectives provided by the BMM, the difference between a goal and an objective lies in the goal's hardness (i.e., whether the goal's satisfaction can be determined). Therefore, from now on, we use the terms "hard goal" and "objective" interchangeably.

Finally, the BMM defines that the performance of a business strategy (means) is measured by the business objectives (ends) that the strategy supports [26]. Thus, the extent to which a software requirement aligns to business strategy depends on the extent to which the requirement contributes to the satisfaction of the strategy's business objectives. Attempting to demonstrate a requirement's strategic alignment to soft goals (e.g., "maximise profit") would be inappropriate, since it would not be possible to describe the extent of the requirement's contribution to the goal. Therefore, when demonstrating strategic alignment, requirements should be related to objectives rather than goals.

IV. RELATED WORK

The following areas of research are related to analysing the strategic alignment of software requirements: (A) Value Based Software Engineering, (B) Goal Oriented Requirements Engineering, (C) Strategic Alignment Approaches, (D) Quantitative Requirements and Metrics, and (E) Requirements Traceability Approaches.

A. Value Based Software Engineering (VBSE)

The VBSE agenda is motivated by observations that most software projects fail because they do not deliver stakeholder value, yet, much software engineering practice is done in a value-neutral setting (e.g., where project cost and schedule is tracked rather than value) [27]. Value Based Requirements Engineering (VBRE) takes the economic value of software systems into perspective through activities such as stakeholder identification, business case analysis, requirements prioritisation, and requirements negotiation [28]. The primary VBRE activities are Business Case Analysis (BCA) and Benefits Realisation Analysis (BRA) [16]. Other VBRE activities such as prioritisation are considered secondary to these, since they depend on benefit estimation [29].

In its simplest form, BCA involves calculating a system's Return on Investment (ROI) - which is the estimated financial gain versus cost, defined in present value. While simple in definition, accurately calculating ROI is complex, since the validity of any concise financial figure depends on assumptions holding true, e.g., that independent variables remain within expected intervals (e.g., time saved is between [x,y]). Estimating benefit involves further intricacies such as uncertainty and the translation of qualitative variables (e.g., the software user's happiness) to quantitative benefits (e.g., sales revenue) - none of which are made explicit by BCA. An advancement from BCA in descriptiveness, e-value modeling seeks to understand the economic value of a system by mapping value exchanges between actors, ultimately leading to financial analysis such as discounted cash flow [30].

However, it does not address how economic value creation is linked to requirements, nor are links between value creation and business strategy attempted.

BRA's fundamental concept is the Results Chain [2], which visually demonstrates traceability between an initiative (i.e., a new software system) and its outcomes (i.e., benefits) using a directed graph, where nodes represent initiatives, outcomes, and assumptions, while edges represent contribution links. BRA's contribution links allow one initiative to spawn multiple outcomes, but the links are not quantitative, e.g., outcome: "reduced time to deliver product" can contribute to outcome: "increased sales" if assumption: "delivery time is an important buying criterion" holds true - but the quantitative relationship between "delivery time" and "sales increase" is not explored. This is problematic when outcomes are business objectives, since their satisfaction depends on the extent that they are contributed to, e.g., in the case of a cost reduction objective, the primary concern is the amount of reduction that is contributed by the actions.

In summary, neither BCA nor BRA estimates the benefit of individual requirements, but rather for whole systems. A similar criticism also applies to the majority of requirements prioritisation methods, as a systematic literature review "found no methods which estimate benefit for [primary] individual requirements" [29], nor were any found which derive the benefits of secondary requirements from their contribution to primary requirements. In this context, primary refers to stakeholder requirements or business objectives while secondary refers to those derived from the primary requirements (e.g., a refined functional requirement).

B. Goal Oriented Requirements Engineering (GORE)

GORE seeks to provide answers to "why?" software functionality should exist. The most well-known GORE methodologies include KAOS [31], * [32] and GRL [33]. Such methodologies produce goal graphs whereby goals at a high level represent the end state that should be achieved and lower level goals represent possible means to that end. A goal graph is traversed upwards in order to understand why a goal should be satisfied, and downwards to understand how that goal could be satisfied. In this context, a requirement is a low level goal where one agent (e.g., a human or a machine) is responsible for its satisfaction. Other related concepts such as resources, beliefs and obstacles are typically related to goals to describe what a goal's satisfaction requires, while agents indicate who is responsible for, dependent on, or wishes for a goal's satisfaction. Relationships between goals are typically represented by means-end links, where optional AND/OR constraints represent alternative options for satisfying a goal. Contribution links are enhanced means-end links, in that they describe the extent to which a goal contributes to the achievement of another. However, "extent" is usually defined in terms of sufficiency and necessity (logic), not as in the quantitative extent of the contribution [34].

i. Goal-Goal Contribution Links

Contribution links are usually annotated with a score or a weight to represent the degree of contribution made by the goal. Three approaches for applying scores to contribution
links in goals graphs are described by Van Lamsweerde [12]:

1. Subjective qualitative scores e.g., --, -, -+-, ++.
2. Subjective quantified scores e.g., -100 to 100.
3. Objective gauge variable (i.e., a measured quantity predicted to be increased, reduced, etc.).

After evaluating the above approaches, Van Lamsweerde concludes that the specification of contribution scores with objective gauge variables (the third option) is the most appropriate for deciding among alternatives, due to its verifiability and rooting in observable phenomena. Of course, the subjective approaches are no doubt quicker to use, but their sole use risks misunderstanding the actual contribution mechanism. A comprehensive comparison of the qualitative contribution reasoning techniques can be found in [35].

Just as objective contribution scoring adds rigour and testability to the task of deciding between alternatives, the same applies to the task of demonstrating alignment to business objectives. Thus, contribution scores should be quantified in terms of the contribution likely to be made to the objective. Our rationale is that, by definition, objectives are quantitatively prescribed, and reasoning qualitatively about degrees of satisfaction of a quantitative target is highly ambiguous. Additionally, this will allow the contribution scores to be verifiable so that they (as predictions) may be later compared to actual results in the evaluation stage of the project. It must be noted here, however, that this option is not without its disadvantages - empirical studies in requirements prioritisation show that practitioners find providing subjective data far easier than objective data [36]. A parallel can be drawn here to the decision analysis field, where inferior (i.e., qualitative) processes have found favour with decision makers because "they do not force you to think very hard" [9].

Van Lamsweerde goes on to explain how alternative goal (i.e., requirement) options can be evaluated by predicting contributions made by goals to soft goals (which define the qualities to be used for comparison) [12]. However, in the prescribed approach, the relationship between the soft goals and the predicted benefit to be gained by their achievement is not made explicit. In other words, the contribution scores are not abstracted to different levels of benefit such that they may eventually relate to business objectives. Each of these potential benefit abstractions require that the metrics used to measure contribution (and satisfaction) are translated (e.g., from time saved to money saved). Furthermore, the expected values allocated to the objective gauge variables are single-point representations of inconstant and variable phenomena. For example, when estimating the number of interactions required to "arrange a meeting via email" — an alternative requirement option taken from the paper’s meeting scheduling system example — a single number does not describe the possible variance, or how that variance can affect the desired end. This is important for predicting strategic alignment, since variance in a requirement’s satisfaction is likely [13], and it will affect the satisfaction of the related business objective(s).

GORE approaches typically describe a goal’s benefit relative to other goals with an importance or weight attribute [12], where importance is a qualitative label (e.g., high, medium, low) and weight is a percentage (where the total of all assigned weights is 100%). Both of these approaches are ambiguous and subjective, and are not traceable to observable benefits, e.g., alignment with business objectives.

A probabilistic layer for quantified goal graphs is proposed in [13] to represent the variance of goal contributions in terms of Probability Density Functions (PDFs). However, effects of the variance on the satisfaction of high-level goals, or business objectives, are not described. To use the example provided in the paper, the effects of an ambulance arriving at a scene within 8, 14, or 16 minutes (i.e., satisfaction of the target exceedingly, completely, or partially) are not described in the context of the benefits of doing so — i.e., to what extent will some problem(s) be affected given these possible goal satisfaction levels. If this is not explored, then it might be that there are no significant benefits to be gained at certain intervals of goal satisfaction levels (note that this point is more significant for non-life-threatening systems). Thus, if a goal is defined with a specific target (e.g., target arrival time) in mind, without the rationale for doing so explored as further goal abstractions, then satisfying that goal may not be worthwhile - "wrong decisions may be taken if they are based on wrong objectives" [13]. Furthermore, probabilistic approaches have limited applications (PDFs are not often available and are time consuming to construct), and do not capture stakeholder "attitude, preference and likings" [15].

ii. The Goal-oriented Requirements Language (GRL)

Given the choice of GORE methodologies, we chose to focus on and adopt GRL [33] for the following reasons:

1. GRL’s diagrammatic notation is well known within the RE community (since it originates from i* [33]).
2. i* (GRL’s primary component) has been shown to be the most suitable for modelling Information System (IS) strategic alignment according to the strategy map concept (GRL not included in review) [37].
3. GRL has an ontology describing its modelling concepts (where others are described informally) [34].
4. GRL was recently made an international standard through ITU specification Z.151 [11].

GRL integrates the core concepts of i* and the NFR Framework [33] (where i* inherits the qualitative goal contribution mechanism from NFR [32], but GRL adds to i* the capability to express contributions quantitatively. Thus, goal contributions in GRL can be specified with either subjective numeric scores (interval [-100,100]) or qualitative labels (one of {−−, −, −+, −++, +++, +++) [33]), i.e., the first and second options outlined in Van Lamsweerde’s paper [12]. For example, a time reduction goal might contribute to an overall-cost saving goal with a contribution weight of 67 out of 100 with positive polarity (+). Such a contribution score is untestable and not grounded by observable phenomena. Moreover it is not refutable, which, according to Jackson [38], means that the description is inadequate because no one can dispute it. The only way such scales could be testable is if the goals were specified with fit criteria (e.g., a cost to be saved), which mapped to the scale, e.g., that they implied percentage satisfaction (which they do not). In which case, a 50/100 contribution might imply that 50% of a £20,000 annual cost saving will be achieved. However, this is only applicable for goals whose satisfaction upper bound is 100% (since the
scale’s upper bound is 100), which is not the case for goals involving increases (e.g., where a means’ contribution to an end exceeds the end’s target level).

Recently, the jUCMNav tool allowed goals to relate to Key Performance Indicators (KPIs) in GRL, in order to map a goal’s satisfaction value to real world numbers [39]. However, subjectivity still exists in goal chains (i.e., >1 link), since KPIs do not affect the way in which goal contribution is specified further up the goal chain (i.e., as low-level benefits are translated to high-level business objectives, e.g., converting time to cost). Also, the interaction between KPIs is not considered, e.g., composition via hierarchy or non-linear causation. Since the publication of our original work, Horkoff et al. have improved GRL’s integration with indicators to consider the hierarchy of KPIs alongside a goal model [40]. However, their approach is concerned with improving Business Intelligence (BI), rather than aligning software requirements to strategic business goals. Thus, several areas are still lacking when applied to our problem. Means are not distinguished from ends (i.e., business objectives and software requirements), making it difficult to know which sets of goals should be aligned, or how those goals should be defined or organised differently. Also, stakeholder utility and confidence through the range of possible goal satisfaction levels (i.e., KPIs in the approach) is not specified – making it hard to know the effects of partial requirement satisfaction, or the credibility of the estimated alignment. Furthermore, non-linear relationships in the associated KPI hierarchy (i.e., diminishing returns in achieving an objective) are not amenable to algebraic description [14] (i.e., “business formulae”, as termed in the approach) – making their definition and communication difficult. Finally, potential fluctuation or uncertainty (i.e., the range of possibilities) in goal contribution is not described, as is done with usage profiles in [41].

As an additional concern, a contribution link is underpinned by assumptions which can either make or break the satisfaction of the end goal. For example, a reduction in task time will only reduce costs if associated costs are actually cut (e.g., by billing work to a different project, or through redundancies). GRL’s belief elements (otherwise known as “argumentation goals”) could be used to express such assumptions in order to provide an integrated view, despite their inferiority in richness to satisfaction arguments [42]. However, in the case of this particular assumption, it seems more semantically appropriate to model it as a necessary action for the end-outcome, just as the BRA’s Results Chain [2] does.

C. Strategic Alignment Approaches

The Balanced Scorecard and Strategy Maps (SMs) approaches [43] offer guidance on formulating and relating business goals to each other under four perspectives: financial, customer, internal processes, learning and growth. In order to improve traceability between these perspectives, SMt* combines SMs with “*” goal models [44]. While this approach does not directly relate to software requirements, goals could be categorised by the four perspectives to ensure coverage.

The most suitable framework for relating software requirements to business strategy is B-SCP [45], due to its tight integration with the OMG’s BMM and the explicit treatment of business strategy that this affords [7]. B-SCP decomposes business strategy towards organisational IT requirements through the various levels of the BMM (i.e., the vision, mission, objective, etc.). However, B-SCP cannot show the extent to which a requirement satisfies a strategy, since no contribution strengths are assigned to links between requirements and the strategy’s objectives. Moreover, B-SCP’s methodology refines business strategy top-down towards IT requirements, which means that completeness of the model is dependent on the completeness of the business strategy, i.e., there is no opportunity to abstract software functionality upwards to propose new business strategy. Additionally, B-SCP does not consider rich GORE concepts (e.g., AND/OR, actors), as found in GRL.

D. Quantitative Requirements and Metrics

The contribution that a requirement’s implementation makes to a business objective depends on the extent of the requirement’s satisfaction (i.e., partially or completely). In order to understand the extent of a requirement’s satisfaction, the desired outcome of the requirement must first be made explicit. Although its practicality is debated [46], it is considered best practice to describe a requirement’s desired outcome using quantitative measures [47]. In [48], Gilb describes the steps that requirements quantification should entail. Firstly, the desired level of achievement should be elicited. Then secondly, the capabilities of the various alternative design solutions should be estimated against that desired level. Finally, the delivered solution should be continuously measured against that desired level. Unfortunately, these steps are rarely followed in practice [47], [48].

As a result of a career training practitioners to quantify requirements, Gilb concludes that there are two main obstacles to quantifying requirements [48]. Primarily, practitioners find defining quantitative scales of measure for a requirement difficult, often believing that it is impossible to quantify all requirements due to their sometimes qualitative nature (guidance on doing so can be found in [51]). Secondly, practitioners encounter difficulty in finding ways of measuring numeric qualities of software which are practical to use (i.e., meters in Plangauge), and at the same time, measure real qualities. Besides, even if a requirement is quantified, its quality is not necessarily improved; a related survey revealed that precisely quantifying requirements can lead to long project delays and increased costs if the quantifications are unrealistic [49]. This is problematic, since it is not straightforward to determine what is realistic with current technology in order to set the desired level of achievement. Despite the difficulties in expressing a requirement’s fit criterion quantitatively, qualitative descriptions (e.g., “good uptime”) are too ambiguous to be useful – in both trying to achieve that requirement, and in analysing the effect of its implementation on the (quantitatively defined) business objectives. The only caveat to this is that qualitative terms such as “good” can be suitable if they have been defined as fuzzy numbers [50].

The Volere [17] template stipulates that a “Fit Criterion” be attached to a requirement in order to make its satisfaction empirically testable (i.e., the first step of Gilb’s requirement quantification steps). Plangauge [51] similarly provides a
template for describing how a requirement’s satisfaction should be tested, and what the result of the test should be. Plangauge’s fit criteria are more descriptive than Voicere’s, since multiple levels of quantitative fit criteria are specified, e.g., for what must be achieved (minimum), what is planned to be achieved (likely), what is wished to be achieved (best case), and what has been achieved in the past (benchmark).

GQM+ Strategies [52] was developed to extend the Goal Question Metrics methodology by providing explicit support for the traceability of software metrics measurement effort at the project level (e.g., measuring the impact that pair programming has on quality) to goals at the business level (e.g., increasing the software user’s satisfaction). In [53], the approach is used to show the alignment of software project goals to high-level business goals using a 2d matrix. The approach’s main benefits are that goals are defined quantitatively using a tried and tested metrics template, and that assumptions which underpin goal to goal contributions are made explicit, much like GRL’s belief element allows for. However, the approach focuses on decisions at the project level, rather than the requirements level (i.e., which projects, rather than requirements, should be implemented?), so is not directly applicable to the problem – a large and variable number of goal abstractions can be required to link a requirement (a means) to a project goal (an end). Additionally, the approach falls short in areas similar to the other methodologies reviewed. Firstly, when a link exists between two goals, the effects of the first goal’s satisfaction on the second goal are not explored. Thus, although each goal has a target satisfaction level (e.g., 5% profit increase), the predicted contributions that its child goals (e.g., software requirements) make toward it are not represented (along with forecast related information such as confidence, evidence, stakeholder agreement, etc.). Therefore, although GQM+Strategies achieves traceability between project goals and business goals, it is not possible to analyse the extent of the strategic alignment of software requirements, since, as aforementioned, requirements often partially satisfy goals [13], i.e., the effect of a requirement’s partial satisfaction is not described in the context of business objectives. Finally, the approach lacks concepts found in GORE which contextualise goals and support decision analysis (e.g., actors, obstacles, AND/OR links).

E. Requirements Traceability Approaches

Several approaches exist which allow means to be traced to ends, typically by constructing a 2d comparison matrix where rows list means and columns list ends. Such traceability allows questions such as “what ends will be affected if this means is affected?” Additionally, it is usually possible to answer the question “how well does this means satisfy the end we want?” One of the most popular tools to trace (and then compare) product features (i.e., means) to customer requirements (i.e., ends) is the House of Quality (HoQ) [54]. Numbers are assigned (e.g., 1-9) to each means-end relationship based on the strengths that the means contributes to the end. A drawback to the HoQ is that the numerical score values used to measure the strength of the contribution are subjective (e.g., strong, medium, weak). Additionally, since the HoQ is constructed using a 2D grid, only two dimensions can be compared in the same grid, i.e., requirements can be related to software goals, but if those software goals are to be related to stakeholder goals or business goals, then additional grids will be required for each extra dimension. If these dimensions are not explored (e.g., if the software project goals are not abstracted to business goals), then the goals that the alternative solutions will be evaluated against may be incorrect (e.g., solution specific or aiming to solve the wrong problem). Despite the drawbacks to using grids, they are argued to be the best means of visually displaying traceability for large numbers of traced entities [55], since they avoid diagrammatic “spaghetti”, and perhaps most importantly, they visualise the lack of traceability with empty cells (e.g., means which do not contribute to an end).

To complement Plangauge, GILB proposes an approach called impact estimation [51], which estimates the impact of alternative system options (i.e., means) against a set of requirements (i.e., ends) using a 2D grid. This approach is very similar to the approach used by Van Lamsweerde to evaluate alternative design options [12], as previously discussed in subsection IV.B.i, and as such, it shares the same problems for application to our problem. The main contribution (related to our problem) of the impact estimation method is that it allows the practitioner to represent their confidence (interval [0,1]) in their prediction of the effect a means has on an end.

V. PROPOSED APPROACH

We propose that GRL goal graphs can be used to demonstrate strategic alignment by linking requirements as tasks (where the task is to implement the requirement) and business objectives as hard goals (where the hard goal brings about some business benefit) with contribution links (where the requirement is the means to the objective’s end). The requirements should be abstracted (asking “why?”) until they link to business objectives. Business objectives then link to higher objectives, until the business strategy is represented.

A. Constructing the Goal Graph

Before looking at how software requirements and business objectives can be connected with goal graphs, we must first explain how we represent the individual concepts.

We define business objectives using an adaptation of the GQM+Strategies formalisation template [56], as in Table 1. Requiring a description of a goal using a metrics template encourages more descriptive goal models, e.g., “improve component lifespan” is defined rather than “improve engine”.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Reduced</th>
<th>Object</th>
<th>GT-BU Fabricated Structures (FS)</th>
<th>Focus</th>
<th>Average Manufacturing Lead Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magnitude</td>
<td>Target: 3 months [reduction]</td>
<td>Threshold: 2 months [reduction]</td>
<td>As-Is: 6 months</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>Average time in months required to have FS parts manufactured from the inception of a new engine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timeframe</td>
<td>1 year after system deployment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope</td>
<td>Gas Turbine Components X,Y &amp; Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Author</td>
<td>John Smith (Component Engineer, GT-BU)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Our modifications to the textual template attempt to improve integration with visual GRL diagrams through:

1. The addition of the most important concept [47] from Plangauge - the scale, which specifies exactly what is to be measured, and the unit of measure.
2. The addition of scale qualifiers [51] to better describe the magnitude, e.g., "threshold" separates acceptable from unacceptable [39]. When we refer to the magnitude of an objective, we refer to the target.
3. The specification of the objective’s activity attribute in the past tense, since objectives represent desired outcomes rather than activities.
4. The removal of the constraints and relations fields - these can be expressed diagrammatically with obstacles and links (e.g., dependencies), respectively.
5. The addition of the author field so that newly proposed objectives can be identified and traced.

For our reference implementation, we use the Volere template to define the attributes of a requirement, primarily because it requires a fit criterion for testing the satisfaction of a requirement. Similarly, an objective can be considered satisfied when the specified magnitude is achieved within the specified timeframe (since benefits are not realised instantly).

Figure 1 shows an example diagram produced following the approach to explore and visualise the strategic alignment of three high-level software requirements.

![Figure 1: Example Strategic Alignment Diagram using GRL](image)

We represent software requirements as GRL tasks (i.e., the task of implementing the requirement) using the naming syntax: "(F/NF)|[Requirement][Fit Criterion]", where "F/NF" is either Functional or Non-Functional. "Requirement" is a short headline version of the requirement description, and "Fit Criterion" is the short-hand version of the metric used to test the requirement’s satisfaction. In order to visualise the objectives (specified by the GQM+Strategies template) in a goal graph, we use GRL hard goals with the naming syntax: "Activity|Object Focus|[Magnitude]".

Soft goal elements (e.g., goals and visions from the BMM) are not defined in the goal graph for the purpose of demonstrating strategic alignment. This is because their satisfaction criterion is undefined and thus immeasurable. Therefore, it is nonsensical to consider that a requirement may either partially or completely satisfy a goal or a vision. However, since objectives exist to quantify goals, and since goals exist to amplify the vision [26], non-weighted traceability between an objective and its goals (and their related visions) should be maintained for posterity and for impact analysis.

A contribution link between a requirement and an objective specifies that the requirement’s satisfaction will achieve some satisfaction of the objective. The extent of the satisfaction is defined by the contribution score specified by the link, and is defined in terms of the objective’s scale (thus making contribution scores testable). A link between two objectives is similar, except that the satisfaction of an objective is measured by its magnitude (target) rather than by a fit criterion. If the contributions of the child elements additively amount to meet the parent element’s specified magnitude, then the model suggests that the parent element will be satisfied.

An "OR" contribution specifies that if there are multiple "OR" links, a decision has to be made about which should be satisfied. An "AND" contribution specifies that all "AND" links are required for the objective to be satisfied. The contribution links (E & F) are of the "AND" type, since both objectives (4 & 5) are required if objective (6) is to be satisfied. Decomposition links can be used to refine a requirement into more specific requirements, much like SysML’s hierarchy link stereotype [22]. The high-level software requirement (3) is decomposed to two lower level requirements (1 & 2) to represent the hierarchy of requirement abstraction. The decomposed requirements (1 & 2) then link to objectives (4 & 5) with contribution links in order to represent what those requirements hope to achieve. The decomposition of requirements continues until the lowest level of requirements are represented. For example, requirement (2) is decomposed to specify which type of analysis should be automated (e.g., structural integrity, cost, etc.). Then, these decompositions contribute to more specific objectives (e.g., "reduce the average time taken to assess structural integrity").
B. Single-Point Goal Graph Quantification

Both requirements and objectives have target levels of satisfaction (i.e., fit criteria and magnitudes) prescribed. This target level is a single point of possible satisfaction, where multiple points refer to satisfaction better or worse than the target level. In Table 2 (which complements Figure 1), we show a sample of quantified contribution scores for this single point of possible satisfaction (i.e., the predicted contribution if the target level of satisfaction is achieved). Note that the numbers are now fictional due to commercial sensitivity.

<table>
<thead>
<tr>
<th>Link</th>
<th>[Contribution] [Activity] [Scale]</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (1→4)</td>
<td>[80%] [Reduction] in [Geometry Creation Time]</td>
<td>1</td>
</tr>
<tr>
<td>D (2→5)</td>
<td>[50%] [Reduction] in [Geometry Analysis Time]</td>
<td>0.75</td>
</tr>
<tr>
<td>E (4→6)</td>
<td>[20%] [Reduction] in [Time Required to Design]</td>
<td>1</td>
</tr>
<tr>
<td>F (5→6)</td>
<td>[13%] [Reduction] in [Time Required to Design]</td>
<td>0.75</td>
</tr>
<tr>
<td>G (6→7)</td>
<td>[3 months] [Reduction] in [Manufacturing Lead Time]</td>
<td>0.75</td>
</tr>
<tr>
<td>H (6→8)</td>
<td>[2 FTEs] [Reduction] in [Human Workload]</td>
<td>1</td>
</tr>
</tbody>
</table>

The quantified contribution for link (C) tells us that objective (4) will be satisfied if requirement (1) is satisfied, since objective (4)’s required magnitude of reduction (80%) will be contributed by the complete satisfaction of requirement (1). Note that where percentages are used as contribution scores on links, this does not infer that a certain percentage of the objective’s magnitude will be achieved (in this case, 80% of 80%). Instead, the focus of the objective (e.g., geometry creation time) will be affected by that percentage in the context of the activity (e.g., a reduction by at least 80%). Contribution links between pairs of objectives are read in the same way; link (E) specifies that the satisfaction of objective (4), determined by its magnitude (target) attribute, will lead to some contribution (at least 20%) toward objective (6).

This abstraction of objectives to higher level objectives allows the benefits to be expressed in terms of high-level business objectives. This is done in order to disambiguate the predicted business value by placing the quantifications into context (i.e., a large saving from a small cost may be less than a small saving from a large cost). It must be noted that a contribution link should represent causation rather than correlation, and thus care should be taken to separate the two as far as possible (guidance on this can be found in [14]).

C. Multi-Point Goal Graph Quantification

Our approach so far represents the contribution that objectiveX makes to objectiveY when objectiveX’s magnitude is completely satisfied (objectiveX is interchangeable with requirementX in this statement). However, it is likely that objectives and requirements will only be partially satisfied, i.e., their required magnitude will likely not be fully achieved. Thus, pessimistic, realistic, and optimistic views (i.e., multiple points of goal satisfaction) of strategic alignment are not currently possible. Also, it is not possible to understand the pareto optimality of software requirements (e.g., where most of the benefit is achieved and where diminishing returns starts to occur). Additionally, the potential for benefit caused by a software feature is finite, e.g., a reduction in Full Time Employees (FTEs) can be gained by task automation – up to a point. Furthermore, conflated goal contribution links whose polarity is mixed can remain hidden until multi-point contribution is modelled. Checking if the relationship between two goals is hump or U-shaped (i.e., not monotonic) will indicate that the causal pathway is more complex than is modelled, and thus the goal graph should be expanded. This separation of causal pathways is advocated both in utility theory for systems engineering in [57], and business system dynamics in [14] - which gives the example: the relationship between “increase pressure to finish work”, positively, and then negatively contributes to the goal “increase employee output”, as fatigue eventually overcomes motivation.

In order to understand the effects of partial satisfaction on the chain of goals, it is important to know the contribution objectiveX makes to objectiveY at various levels of objectiveX’s satisfaction. This is represented by defining a table function [14], i.e., pairs of objectiveX and objectiveY values, together with a chosen interpolation method (linear, step-after, cardinal, monotone, etc.). Table functions are used since analytic (i.e., algebraic) functions are difficult to design, experiment with, and communicate to stakeholders when used to model non-linear relationships [14]. For simple linear relationships (e.g., converting between units on an infinite scale), algebraic functions will likely be quicker to define. Table functions should span the worst to best-case range for an objective. If the value of objectiveX is outside the function’s domain, i.e., objectiveX’s value is extreme, then ideally the table function should be updated, since the worst or best-case points are no longer representative. Otherwise, values lower or higher than the function’s domain could be mapped to the function’s minimum and maximum values. Alternatively, the slope of the last two points could support extrapolation.

The effect on the number of "FTEs required for GT-BU Fabricated Structure (FS) Design" when the "Avg. Time Required to Design FSs" is reduced.

To illustrate a multi-point quantified contribution link, Figure 3 visualises link (H), which is comprised of four pairs of values, and, in this case, step-after interpolation to represent integer increments (in other settings, linear interpolation and rounding may be more fitting). In this contribution link,
extreme values of objective (6) are mapped to the minimum and maximum data point specified by the table function, in order to represent finite benefit realisation. Improvements to the reusability and robustness of the relationship, currently in the form of \( Y = f(X) \), could be made by normalising the function such that the input and output of the function are dimensionless, i.e., independent of the unit of measurement used (e.g., to define time or human resource usage). Guidance on constructing non-linear functions can be found in [14].

The visualisation appropriate to depict a contribution link depends on the type of numerical data (i.e., discrete or continuous) used by an objective's scale or a requirement's fit criterion. For functional requirements, a bar chart should be used, since they have two states (i.e., implemented or not), whereas non-functional requirements should be represented using line (xy) charts, since they have infinite states of satisfaction. Note that the green lines on the axes represent the magnitudes (targets) required by the respective objectives, as specified by the goal formalisation template (as in Table 1).

**D. Describing Confidence in Quantifications**

Contribution links in goal graphs are predictions of a causality relationship between two goals. Epistemological uncertainty (caused by a lack of knowledge) about a contribution link therefore must exist to some degree, since we cannot have perfect knowledge about future events. Aleatoric uncertainty (not caused by a lack of knowledge) also exists, in that a requirement's contribution to an objective (i.e., its benefit) depends on the system's environment (i.e., context or scenario of use) [47]. This could be represented by specifying contribution scores for each of these environments, as we exemplify in [41] (usage profiles). However, in this paper, we are interested in the average contribution made over all usage profiles (i.e., considering all likely types/contexts of use). Before we look to describe confidence in goal contribution links, we must first distinguish **certainty** from **confidence**.

When predicting unknown quantities, uncertainty refers to beliefs about possible values for the unknown quantity, while confidence refers to the belief that a given predicted value is correct [58]. For example, with reference to contribution link (H), uncertainty represents the range of possible values of FTE reduction (e.g., an interval [0, maxWorkloadInfTEs]) that could reasonably occur given a reduction in design time of 33%, i.e., the satisfaction of objective (8). In terms of Figure 3, uncertainty would affect the thickness of the line (i.e., lack of precision) used to represent the causation. Confidence, on the other hand, represents the belief that the chosen prediction (e.g., 2 FTEs) is the correct one. Thus a stakeholder's confidence is influenced by the salient factors that they believe contribute to the correctness of their prediction, while a stakeholder's uncertainty is influenced by the number of different prediction options that could be correct [58].

In this paper, we focus on confidence, since empirical studies have shown that while practitioners can judge which of their predictions are more uncertain, they find quantifying the uncertainty interval difficult [59]. However, if a stakeholder were reluctant to provide a single value to quantify the contribution, the contribution could be specified in more uncertain terms, such as: \( 2 \pm 1 \) FTEs for link (H), i.e., an **interval estimate** [60]. It is important that the range is restricted as far as possible to avoid ambiguity in the contribution description, since the utility of a prediction is diminished by imprecision. If the range of uncertainty is wide, it should be expressed with a Probability Density Function (PDF) to show which points in the range are more or less certain [13]. For a single-point contribution relationship (where it is assumed that the target of the first objective will be met), a PDF would describe the distribution of belief over the range of possible values for objective Y, given a specific value of objective X (i.e., objective X's target). However, many values of objective X (i.e., the x-axis in Figure 3) are possible, leading to many possible PDFs to describe - reducing the usability of the approach. Thus instead, stakeholders should be encouraged to specify a single value of objective Y that they are the most confident of, i.e., a **point estimate** [60], as in Table 2.

In the decision analysis field, it is well recognised that representing confidence is essential in determining optimal decisions, especially where a choice has to be made between two options which seem to provide similar benefits [9]. Furthermore, the description of confidence will indicate the risk that the modelled strategic alignment may not occur in practice. The confidence level representation concept we adopt is similar to that used by Glib for impact estimation [51], so, in Table 3, we enumerate confidence levels using a similar scale. Mapping textual descriptions to confidence values (interval [0,1]) allows stakeholders to more easily select a value based on the quality of the supporting evidence (i.e., salient factors).

<table>
<thead>
<tr>
<th>Confidence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>Poor credibility, no supporting evidence or calculations, high doubt about capability</td>
</tr>
<tr>
<td>0.5</td>
<td>Average credibility, no evidence but reliable calculations, some doubt about capability</td>
</tr>
<tr>
<td>0.75</td>
<td>Great credibility, reliable secondary sources of evidence, small doubt about capability</td>
</tr>
<tr>
<td>1</td>
<td>Perfect credibility, multiple primary sources of evidence, no doubt about capability</td>
</tr>
</tbody>
</table>

Basic confidence adjustment can be performed by multiplying contributions by their associated confidence level so that users are reminded of the impact confidence has on predictions, as in [51]. For example, when confidence levels are taken into consideration in Table 2, the satisfaction of requirement (1) still leads to the full satisfaction of objective (4). However, when confidence levels are considered for links (E & F), the satisfaction of objective (6) is in doubt, since \( (20^*1) + (13^*0.75) \) is less than the 33% required by the objective's magnitude attribute. Adjusting contributions to account for confidence in this way is similar to calculating the expected value of a random variable. However, since the mapping between the textual statements and the numbering in Table 3 (adapted from [51]) is not grounded by evidence or heuristics, a contribution score which is adjusted for confidence using them should not be treated as an expected value, but rather as an indication of the effects of confidence. If we wanted to better approximate the expected value, a number based on probability should be used to represent confidence [58], i.e., the answer to such a question: if the re-
requirement were implemented a large number of times, what percentage of those times would (at least) the stated contribution be contributed? The answer to such a question depends on the experiences of the stakeholders in implementing similar requirements in similar projects in a similar environment. However, similarity in this sense is difficult to achieve, since there are many socio-technical variables that can affect the benefits realised by a software project or a particular feature.

Additional confidence levels could be associated to the user’s predictions to represent how qualified that user is at predicting contribution scores. For example, someone who has implemented similar systems should be able to provide more accurate predictions than someone who has not. The accuracy of a person’s previous predictions (i.e., their credibility) could also be considered in order to improve the reliability of the predictions (i.e., calibrated confidence levels).

E. Describing the Utility of a Goal’s Satisfaction

One important value consideration is so far, untreated: "what is the benefit in achieving a root goal to various degrees of satisfaction?" i.e., business objectives that do not contribute to other business objectives, such as objective (12). Root objectives exist when the business has not defined any objectives higher than the objective, and where it would not make sense for them to have done so. To address this, we map various levels of a root goal’s satisfaction to degrees of utility [9], whereby various levels of “goodness” can be achieved. For example, referring to objective (12), various levels of component lifespan improvement map to utility values (interval [0,1]). This allows the representation of non-linear relationships between component lifespan improvement and the associated benefit; perhaps after a 60% improvement on the average component’s lifespan, there is no more benefit to be gained since the engine would be retired before the component would fail. Thus, the utility of a 60% improvement would peak at 1. The concept of utility is both subjective and specific to the stakeholder who assigned it. However, capturing it will explain the criticality of a root goal’s satisfaction criterion, and differences in utility assignment between stakeholders will be made apparent for conflict resolution before the requirement is implemented.

Note that the maximum utility of some goal satisfaction is defined in isolation from other goals. That is to say that the maximum utility value (i.e., 1) should be defined for each root goal, and then weights can be assigned to those root goals to determine the relative utility of some goal satisfaction, in the context of the system-to-be as a whole. This is done in order to decide on the relative importance of root goals, as in [61]. Pairwise comparison or balance beam diagrams can be used to decide on, and refine the weights [57].

F. Describing & Monitoring As-Is Values for Goals

Wherever the magnitude attribute of an objective (and related contribution scores) is/are specified a percentage, it is especially important that the objective’s as-is value is described. Otherwise, it is not possible to later verify that the magnitude (i.e., change) has happened. These values can then be recorded over time in order to evaluate the system (validation) and provide a benchmark for future improvement. Furthermore, prescribed goal satisfaction levels and predicted goal contribution levels, in current and future projects, can then be made more realistic through a feedback mechanism.

G. Describing Assumptions or Necessary Goals

When a contribution link is made between two goals, there may be an assumption that some other necessary action will occur which enables the contribution. For example, Figure 1’s contribution link (K) is underpinned by the assumption that design costs can be reduced through employee time reduction. While this may seem trivial to highlight, the actual cost reducing mechanism (perhaps redundancy) may be a thorny issue, and should be communicated as early as possible for conflict resolution. To describe this assumption, a new task could be added as a decomposition of objective (11), since GRL’s decomposition links represent necessity (while contribution links represent sufficiency and polarity, i.e., +ive or -ive). Assumptions made in the calculation of contribution scores should also be made explicit, e.g., in link (H), that an FTE is 40 productive hours per week. This could be represented with a GRL belief node connected to link (H).

H. Intended Context of Use

We suggest that this approach should be performed after the high-level requirements have been elicited, so that resources are not wasted eliciting lower level requirements that do not align well to business strategy. It is especially important that the strategic alignment of solution oriented requirements (i.e., those specified for the machine [38]) is explored, since they do not explain the problem to be solved.

It is important to note that software engineers and business analysts may not know the objectives (or the goals and visions, for that matter) at different levels of the business (i.e., the project, the business unit, the department, the overall business, etc.). Therefore, managers should work with stakeholders to define the business objectives before the requirements can be abstracted toward them. Indeed, it is likely that some software requirements will be abstracted toward business objectives that were not previously elicited.

Practitioners may find difficulty in quantifying the benefits of requirements, especially for non-functional requirements where the subject may be intangible, however, proxy indicators can usually be identified relatively easily (e.g., by polling customers to quantify customer satisfaction using a likert scale) [51]. Furthermore, where stakeholders are unable to quantitatively explain the causal relationship between a requirement and higher goals (either in the description of the numeric scales or in their values), the risk that the contribution may not occur as expected will have been indicated.

While we have focused on the benefits of software requirements, both sides of the value equation need to be considered (i.e., costs). Effort estimation models such as COCOMO [62] would be useful in predicting the development cost of a requirement. The effort required by the end users to use an implemented requirement should also be considered.

I. Tool Support

Tool support (GoalViz) has been developed (free to download at [63]) to support the approach through:
• Input support for requirements, objectives, and contribution data (with graphical function input).
• Automatic diagram drawing to focus the user on the approach and data rather than the graph layout.
• Project libraries to facilitate learning about the contributions predicted in previous projects to improve future quantification of confidence assignment.
• Automatic evaluation and summarisation of chains of links to enable efficient understanding.
• What-if analysis allowing comparison of outcomes for different inputs where there is some uncertainty.

VI. CONCLUSION AND FUTURE WORK

The presented approach facilitates the disambiguation of a requirement’s business value through the enrichment of contribution links in a goal graph. The approach is descriptive (a goal is abstracted to describe the underlying problem), prescriptive (a certain amount of goal satisfaction is required), and predictive (a quantitative goal contribution score predicts how much of the prescription will be achieved by the means). This paper’s unique contribution includes:

• We have argued that the strategic alignment of software requirements depends on the contribution they make to business objectives, and since they are quantitative in nature, reasoning about the contribution made toward them should also be quantitative;
• We have argued that since strategic alignment is based on predictions of benefit, confidence (and sometimes uncertainty) should be made explicit;
• We have shown that the non-linear dynamics of goal contribution links can be explored as quantitative causative relationship (defined with table functions) through more than one level of goal abstraction, in order to understand the effects of partial requirement satisfaction on high-level goals.

Future work will evaluate this approach (and those related to it) against required capabilities elicited from our industrial partners. We have outlined two case studies within different industrial settings, such that the benefits and challenges can be evaluated in the context of a range of domains. Feedback resulting from the evaluations in industry will be used to improve the approach and the tool. Planned investigations into optimising the utility and the usability of the approach include empirically evaluating the:

• Extent to which stakeholder utility functions for a goal’s satisfaction can be aggregated to represent the preferences and uncertainty of a collective;
• Optimal representation of uncertainty, confidence, and credibility in the causal relationships;
• Optimal way (especially with regards to reusability) to specify the causal relationship between two variables (i.e., gauge variables, KPIs, or goal satisfaction levels), e.g., with causal loop diagrams and dimensionless table functions using Vensim® [14], or specifying “business formulae” as in [40];
• Optimal way to maintain traceability between requirements and design artefacts, perhaps through SysML Requirements [22] and a GRL UML profile.

ACKNOWLEDGEMENTS

The authors wish to thank Ralph Boyce from Rolls-Royce for his valued participation and feedback in this project, and also Rolls-Royce for granting permission to publish.

REFERENCES


Analyzing the Assumed Benefits of Software Requirements

Richard Ellis-Braithwaite
Loughborough University, United Kingdom
r.d.j.ellis-braithwaite@lboro.ac.uk

Abstract. Often during the requirements engineering (RE) process, the value of a requirement is assessed, e.g., in requirement prioritisation, release planning, and trade-off analysis. In order to support these activities, this research evaluates Goal Oriented Requirements Engineering (GORE) methods for the description of a requirement’s value. Specifically, we investigate the goal-to-goal contribution relationship for its ability to demonstrate the value of a requirement, and propose that it is enriched with concepts such as correlation, confidence, and utility.

Keywords: software requirements, assumptions, benefits, strategic alignment

1 Problem Statement

There is often a “field-of-dreams” assumption that once software is built to the specified requirements, benefit will come [1]. The fact that there is little correlation between a company’s level of IT investment and its profitability or market value, leading to the so-called “information paradox” [2], highlights the dangers of this assumption. Therefore, stakeholders responsible for a software project’s funding need to be able to demonstrate that the software will be beneficial. Furthermore, practitioners performing RE processes where the benefit of a requirement is assessed (e.g., in prioritisation) need to know how benefit is defined by the stakeholders, and then how the requirements will contribute to it. With this in mind, this research aims to explore, improve, and evaluate methodologies for analysing the assumed benefits of requirements, and thereby, the alignment of those requirements to business strategy. Through evaluation of the methods in industrial projects, we aim to optimise the cost to benefit ratio of their application through methodology improvements, guidelines, and tool support.

The following research questions were formulated in response to problems faced by our industry partners, with the overall goal to improve decision-making in the RE process via better communication and analysis of assumed benefit:

RQ1. What evidence exists to show that implemented requirements (i.e., software features and qualities) are not always beneficial?

RQ2. What is an appropriate approach for modelling the assumed benefits of software requirements?

RQ3. What aspects of the resulting benefit model are important for analysing the strategic alignment of software requirements?

RQ4. What are the quality characteristics of such models, and what challenges preclude them?

RQ5. How can a supporting tool address the challenges elicited from RQ4?
2 Motivation

The oft-cited, but ageing CHAOS report [3] suggests that two thirds of functional requirements which are specified before implementation are never or rarely used after their implementation. This claim is supported by independent surveys, for example [4], which found that only 27% of the functionality in word processing software is ever used. Since stakeholders are often “motivate[d] to brainstorm requirements which they think that they just might need at some point” [5], it should be no surprise that some requirements lack pertinence, and as a result, deplete development resources.

Pertinence also affects the quality of non-functional requirements (NFR’s). For example, a reliability requirement stipulating a certain level of service uptime should be the result of a trade-off made between two or more conflicting stakeholder goals, e.g., “maximise service availability” and “minimise infrastructure costs”. These trade-offs aim to maximise the utility of the software by optimising the associated cost-benefit ratios and acceptable risk levels [6]. Unfortunately, it has been found that stakeholders are rarely the source of NFR’s, since “architects consider themselves to be the real experts when it comes to defining efficiency, reliability, and other similar aspects” [7]. On the contrary, the RE activity is primarily concerned with the description of the application domain [8] - the stakeholder’s specialism, rather than the machine, which is the architect’s specialism. In order to perform a successful trade-off, the rationale behind each goal is required, and without stakeholder involvement, over/under specification of NFR’s will likely occur as a result of developer assumptions, ultimately leading to increased costs, delays, or in extreme cases, project failure [9]. Thus, for such consequences to be avoided, developers and stakeholders need to be able to comprehend the effects of a requirement’s implementation on each other’s goals, since the quality of any decision is underpinned by the information available to support it [10].

Numerous surveys blame the majority of software project failures, including poor return on investment (ROI), on inadequate RE [3, 11] - or more specifically, on poor stakeholder communication and incorrect assumptions [12]. Framing the problem in the context of these failure factors, we wish to minimise assumptions made about the benefits that stakeholders expect, by communicating those expectations to developers. Regardless of the cause, as a result of IT’s poor ROI, IT-business strategy alignment has been the top ranking concern of business executives for the last two decades [13].

3 Related Work

The value based software engineering (VBSE) agenda [14] is motivated by the observation that most software projects fail because they don’t deliver stakeholder value, yet, much software engineering practice is done in a value-neutral setting (e.g., where project cost and schedule is tracked rather than stakeholder value). Value-based requirements engineering (VBRE) takes the economic value of IT products into perspective through stakeholder identification, business case analysis, requirements prioritisation, and negotiation [15]. The primary VBRE methods are Business Case Analysis (BCA) and Benefits Realization Analysis (BRA) [16]. We consider other VBRE processes (e.g., prioritisation) as secondary, since they depend on benefit estimation.

In its simplest form, BCA involves calculating a system’s ROI (financial benefits versus costs, in present value). An advancement from BCA, e-value modelling seeks to understand the economic value of a system by mapping value exchanges between actors, ultimately leading to financial analysis such as discounted cash flow [17]. Howev-
er, these approaches are complex in their application, since the validity of any concise financial figure depends on assumptions holding true, e.g., that independent variables remain within expected ranges. Estimating benefit involves further intricacies such as uncertainty, and the translation of qualitative variables (e.g., software user happiness) to quantitative benefits (e.g., sales revenue) - none of which are made explicit by BCA.

BRA's fundamental concept is the Results Chain [18], which visually demonstrates traceability between an initiative (i.e., a software system) and its outcomes (i.e., benefits) using a directed graph, where nodes represent initiatives, outcomes, and assumptions, and edges represent contribution links. BRA's contribution links allow one initiative to achieve multiple outcomes, but the links are not quantitative, e.g., outcome: “reduced time to deliver product” can contribute to outcome: “increased sales” if assumption: “delivery time is an important buying criterion” holds true – but the quantitative relationship between “product delivery time” and “sales increase” is not explored. This poses a problem when outcomes are business objectives, since the satisfaction of an objective depends on the extent that it is contributed to, e.g., in the case of a cost reduction objective, the extent is the amount of reduction that will be contributed [19].

Goal Oriented Requirements Engineering (GORE) methods are capable of demonstrating alignment between software requirements and goals with AND/OR goal graphs [20]. Goal graphs ensure the pertinence of software requirements [20], since requirements must trace to a more abstract goal to explain the rationale for the requirement's implementation. Additionally, goal graphs improve communication between stakeholders since requirements are restated at various levels of abstraction (through goals), thereby bridging the gap between technically minded developers and application-domain focused stakeholders. Quantified goal graphs are used in [20] to model how far one goal contributes to its parent goals. However, the approach does not translate contribution scores toward various levels of goal abstraction, and therefore does not place the benefit contributed by a requirement into context, e.g., that a large saving may only be derived from a small cost [19]. A probabilistic layer for quantified goal graphs is proposed in [21] to reason about the probability that a goal’s more abstract parent goals will be satisfied. However, this approach is time consuming, has limited applications, and does not capture stakeholder “attitude, preference and likings” [22].

Singh and Woo review the IT-business alignment literature within the RE field [23], and conclude that the majority of frameworks do not address business strategy or value analysis. The Strategic Alignment Model [24] is proposed as a theoretical framework for conceptualising IT-business alignment, but it is not taken beyond the conceptual level, and thus does not consider traceability to system requirements [25]. The OMG's standardised Business Motivation Model (BMM) [26] states that a set of quantified business objectives forms a business strategy, and that satisfaction of the objectives satisfies the strategy. Several methodologies have used the BMM to show the strategic alignment of software requirements. For example, in [27], OMG's SysML requirements metamodel is extended to establish traceability to the “tactic” concept from the BMM. Similarly, in [25], the BMM is used to decompose business strategy down to software requirements (i.e., from vision statements to tasks) which will satisfy the strategy. However, much like the BRA, neither approach takes a quantitative approach to demonstrating strategic alignment, yet strategic alignment depends on the extent of an objective's satisfaction, which is measured quantitatively.
4 Proposed Solution

The foundations for our approach to answering RQ2 are introduced in [19]. A brief summary is that the benefits of software requirements are stated at various levels of abstraction using goal graphs (where a benefit is the advantage gained by treating a problem and where goals are inverted problems). Goal graphs are used because they are well suited for visualising abstraction and refinement (hierarchies between parent/child goals), and can also visualise dependencies and cause-effect relationships [28].

A more detailed summary is that GRL goal graphs [29] are used to relate software requirements (represented by GRL task elements) to business objectives (represented by GRL hard goal elements) through GRL contribution links. Software requirements are represented by GRL tasks, in that it is the task of implementing the requirement that should satisfy some business objective. Business objectives are specified using the GQM+Strategies template [30], such that objectives can be considered as GRL hard goals (i.e., the objective’s required magnitude is set). The proposed approach is: a) prescriptive in that some amount of business objective satisfaction is prescribed, b) descriptive in that the problem to be solved by the objectives is contextualised by various levels of goal abstraction, and c) predictive in that the contributions made by requirements or objectives to higher-level business objectives are quantitatively estimated.

In order to illustrate the approach’s application, we refer to a software project that the authors were involved with. The software manages and schedules media files for digital signage (advertising). Fig. 1 shows an excerpt of a goal graph in the context of this project (in accordance with [19]), which explores some assumed benefits of a proposed functional requirement. A reader unfamiliar with GRL can find guidance in [29].

![Fig. 1. Example goal graph showing the abstracted benefits of a proposed functional requirement](image1)

![Fig. 2. Contribution of [G1] → [G2]](image2)

When it comes to answering RQ3, we are most interested in the contribution links between software requirements and business objectives, since they represent the alignment. Specifically, we are interested in describing GoalX to GoalY contribution
such that various levels of satisfaction of GoalX are mapped to various levels of satisfaction of GoalY. We are most interested in causal relationships between GoalX and GoalY, but in reality correlation and causation are difficult to distinguish in sociotechnical systems [31]. We also take into account the effects of the system’s various usage profiles on goal contribution, as shown in Fig. 2 by the y-axis grouping (the “Promo” group represents a promotional period, such as a seasonal holiday).

For functional requirements, the contribution data is discrete, since there are only two states to capture - when the requirement is, or is not implemented. Thus, in Fig. 2, we refer to [T2] as the As-Is state to represent when the requirement is not implemented, and [T1] as the To-Be state to represent when the requirement is implemented. For non-functional requirements, the contribution data is continuous, since when properly specified, their values are numerical (e.g., an uptime requirement will have various possible states of satisfaction: 99.99%, 99.98%, etc.). Thus, if [T1] were non-functional, e.g., “the maximum time an expired media item should be displayed for is 5 minutes”, then Fig. 2 would be an XY graph rather than a bar graph, with x-axis values specified in minutes and y-axis values derived from the goal that it contributes to. Note that soft goals are not used, since business objectives are defined quantitatively according to the BMM, and thus qualitative contribution to quantitative goals is ambiguous.

Once the first contribution link has been described, we move up the goal graph to describe the next contribution link, as in Fig. 3, to provide context to the benefits (in Fig. 3, we show that the menial work is only partly comprised of old media removal). Enriching the contribution links with this information will allow “what-if?” analysis through interpolation and contribution propagation. Additional benefits include better elicitation of goal tolerance levels and improved clarification of the goal’s criticality, i.e., the extent to which a goal’s satisfaction criteria can be stretched (e.g., relaxing a non-functional requirement) without causing failure of the goals that it contributes to.

To enrich the contribution link further, we make confidence explicit, that is, when a practitioner describes the correlation between two goals, their confidence in their descriptions will vary depending on their expertise and previous experience. Thus, capturing confidence (e.g., between 0 and 1) for each data point in a contribution link will allow decisions to be made in consideration of the assigner’s uncertainty (lack of knowledge), previous accuracy in their confidence assignments, and their risk preference. Alternative approaches to representing confidence will be evaluated (e.g., enumerated estimate points {worst-case, likely and best-case}, intervals, and probability distributions).

Finally, root goals (those which do not contribute to other goals) are mapped to utility [10], where various levels of goal satisfaction result in various levels of “goodness”. For example, referring to the root goal [G4] in Fig. 1, the various levels of staff motivation (e.g., measured on a likert scale between 0 and 5) would map to utility values (e.g., between 0 and 1). This will allow non-linear relationships between motivation and the utility of that motivation to be represented (e.g., that the difference between 0 and 1 on the motivation scale is bigger than 4 and 5). The concept of utility is both subjective and specific to the utility assigner. However, capturing it will explain the criticality of a root goal’s satisfaction criteria, and any differences in utility assignment between stakeholders will be made apparent before the requirement is implemented. Thus, for this to be useful, the stakeholder’s utility preferences must be communicated to each other for conflict resolution. Then, the stakeholder’s utility functions can be aggregated in order to improve their integrity, as in the “wisdom of the crowd” theory [32].

RQ4 examines the qualities of a model used for the analysis of strategic alignment. The elicited qualities are much like the “completeness” quality of a requirements doc-
ament in that they are aspirational (their complete satisfaction is not expected). So far, we have elicited three core qualities from the application of our approach:

- **Determinism** – a goal graph produced by one practitioner relating a requirement to an organisation’s goals should be similar to one produced by another. This can be supported by defining goals with metrics formalisation templates.

- **Transparency** – a goal graph should be self-explanatory rather than reliant on assumed widespread knowledge of what a goal means or why it matters. This can be achieved with goal abstraction to place goals into context.

- **Reusability** – the applicability of goal definitions (and their associated contribution links) to future projects should be as wide as possible. An example of poor reusability would be describing a contribution link relatively (e.g., “10% user task time reduction leads to 20% task cost reduction”) without providing the absolute figures (e.g., $x$ dollars were saved by reducing $y$ hours).

The primary challenge to the first two qualities is that objective data is far rarer than subjective data. Decision makers have found favour with inferior processes (e.g., qualitative goal contribution scoring) because they do not force you to think very hard [10]. For example, most managers will have the opinion that reducing menial work will improve employee motivation. Thus, assigning a “+” to that contribution link is easy, but understanding the extent to which menial work affects motivation requires a survey, since it might be the case that employees are unaffected by, or even enjoy menial work. Future work will attempt to understand which goal contribution links are in most need of analysis, since it is recognised that practitioner time is finite.

RQ5 looks at how far a tool can support the application of the proposed approach. A tool is currently under development¹, whose current features were derived from inefficiencies encountered whilst implementing the approach. For example, automatic goal graph drawing resulted from the observation that drawing large goal graphs is time consuming due to the number of edge collisions that occur. We plan to extend the tool’s functionality with features such as goal similarity analysis (duplicate detection) in order to support the reuse of data from previous projects (e.g., benefits of a feature).

5 Research Method, Progress & Novelty

This PhD project is in collaboration with two industrial partners (LSC Group and Rolls-Royce)². One industrial partner will provide case studies for software developed for their own organisation, while the other will provide case studies for software developed for an external organisation. The research approach adopted for this thesis is based on the experimental software engineering paradigm [33]. Firstly, a problem was identified with the help of our industrial partners. Structured interviews and questionnaires were then used to investigate the problem, which complements the motivation identified from the literature (briefly discussed in Section 2). Then, the scientific problem was defined in the format of research questions. After that, a solution idea was formed following a systematic literature review. A prototype tool was then developed to make the required data’s capture, representation, and analysis possible so that the solution idea can be improved through feedback. The solution idea will then be validated against the scientific and the practical problem using case studies for evaluation.

¹ The prototype tool is available to download at http://www.goalviz.info/REFSQ-DS/
² The author wishes to thank Dr. Tim King, Dr. Badr Haque & Ralph Boyce for their support as industrial supervisors, and Dr. Russell Lock and Prof. Ray Dawson as academic supervisors.
This PhD project is half way through its three year duration, and an initial solution has been outlined. The remainder of the project will focus on case study research to evaluate the effectiveness of the approach compared to the current state of the art. Initial feedback from the tool’s evaluation with practitioners is promising. Stakeholders, especially business managers, are attracted to the ability to understand technical software requirements in terms of the business objectives that they are familiar with. Feedback from developers has been more critical, since they are evaluated on the quality and timeliness of their programming, rather than on the value aspects of the software. Additionally, the numerical aspect of the approach has been off-putting to some.

As for novelty, we are not aware of an approach that considers the benefits of a software requirement as a chain of quantified goal abstractions. In particular, we are not aware of an approach that attempts to forecast the effect of a goal’s satisfaction on its parent goal(s) at:

a) varying levels of goal satisfaction extent (explaining the effects of partial/full requirement satisfaction on multiple levels of goal abstraction);
b) varying levels of software usage (explaining the different profiles of software usage that can affect a requirement’s contribution to goals);
c) varying levels of stakeholder confidence (explaining the extent to which a requirement’s satisfaction may not contribute to a goal as specified);
d) varying levels of stakeholder utility (explaining the non-linear relationships between the extent of a goal’s satisfaction and the utility gained);
e) varying levels of stakeholder agreement (explaining the variance between the stakeholder’s estimates about the benefits that will be realised).

References

Analysing the Assumed Benefits of Software Requirements (REFSQ)

Richard Ellis-Braithwaite
cordje2@lboro.ac.uk

Introduction: The Underlying Context (1/3)

• Who’s involved?
  PhD project in collaboration with industrial partners for problem identification and solution validation:
  • **** Civil Aero (focus on bespoke software for internal use)
  • **** Technical Consulting (focus on COTS software deployment/customisation for external use [mostly by the Ministry of Defence])

Introduction: The Underlying Context (2/3)

• What are we looking at?
The problem was first identified while working at *** (and then later found to exist in the MOD through working with ***):
  Software projects are often successful in that their requirements are usually met. However, the software rarely adds the value that was initially expected.

Introduction: The Underlying Context (3/3)

• Which leads us to ask:
  “How can we model the value of software before it exists... so that its value can be optimised... and so that stakeholder expectations can be sanitised?”

Introduction: The Research Questions

• RQ1. What evidence exists to show that implemented requirements are not always beneficial.
• RQ2. What is an appropriate approach for modelling the assumed benefits of software requirements?
• RQ3. What aspects of the resulting benefit model are important for analysing the strategic alignment of software requirements?
• RQ4. What are the quality characteristics of such models, and what challenges preclude them?
• RQ5. How can a supporting tool address the challenges elicited from RQ4?

Motivation from the Literature

Summary: lots of waste occurs in software development/acquisition, and it’s not just the customer who pays for it! Satisfied requirement != +value.

<table>
<thead>
<tr>
<th>Key Message</th>
<th>64% of delivered software functionality is never/rarely used (46%,19% respectively)</th>
<th>73% of COTS software is never used, which leads to &quot;bloated software&quot; that is hard to maintain and to use</th>
<th>There is little to no correlation between a company’s level of IT investment and its profitability</th>
</tr>
</thead>
</table>
Motivation from Rolls-Royce - Context

Rolls-Royce "Automated Design Make" Business Unit

- Requirements phase is long (~1 year). Lots of stakeholders from different business units (aerodynamic performance, thermal, noise, etc.) all need to provide input (e.g., design rules), which is a scheduling problem in itself.

Motivation from Rolls-Royce

- Business stakeholders are not involved in requirements/design decisions because they don’t understand the engineering domain - despite the fact that decisions made there impact the satisfaction of their objectives.
- “Why do we need this function, and why should its output be this precise?”
  - “We’ve always done it like that, and Bob (who knew why we did this) left 2 years ago.” – Actual Response [Knowledge Mgmt. Problem]
- In a sample project, of 41 High Level Requirements, 41 were “Must Have” – stakeholders believe their requirements are the most important because they only know their domain (they are specialists). Due to the fragmentation of the organisation, software systems are often considered in the scope of the local supply chain unit or business unit.
- Some projects end up years late (and then are redundant) as a result of the high number of requirements and their acquisition times.

We Know What’s Below a Requirement…

Levels of Abstraction in Model Driven Architecture

Alignment: System Goals & Business Goals

OMG’s Business Motivation Model (bounded by the vision of the overall business)

System Goal Model (bounded by the ability of the system’s agents to influence the goals) (image adapted from van Lamsweerde)

But What’s Above It?

“If You Don’t Know Where You’re Going, Any Road Will Get You There.”
– The Cheshire Cat, Alice In Wonderland

Current State of the Art

- Goal Oriented Requirements Engineering (GORE)
  - KAOS and GRL methods can show alignment from one goal (e.g., a functional requirement) to higher abstract goals (e.g., a stakeholder goal) using goal graphs. However, goal-goal contribution is not considered in terms of the effects it has all the way up the goal chain.
- B-SCP Requirement Strategic Alignment Framework
  - Aligns software requirements to elements from the OMG’s BMM (e.g., tactics), but does not use contribution weights of any kind (what is the extent of the alignment?), does not allow for partial goal satisfaction.
- House of Quality Diagram (QFD)
  - Analyses the extent that means (e.g., features) satisfies ends (e.g., stakeholder goals), but does not use application domain metrics to explain the contribution made (not verifiable), does not place the goals into context, does not consider confidence despite the uncertainty involved.
Our Approach: A Brief Summary

• We chose the GRL language to represent goal graphs, which connect software goals to business goals. GRL is used primarily because it is an international standard (Z.151) with a pre-defined metamodel.
• We specify goal-goal contribution for at least one level of satisfaction (e.g., what is likely to be contributed by goalX's satisfaction at goalX.thresholdLevel, to GoalY's satisfaction).
  - Where goalX has levels e.g., {worstCaseLevel and bestCaseLevel}
• We enrich the GRL metamodel with:
  - a metrics specification framework (GQM+Strategies) to define the goals such that they become verifiable and objective
  - a confidence assessment framework (Gilb’s Impact Analysis)

Our Method: An Example Alignment Diagram

If this point (or any) changes, the entire chain of cause/effect does too. (Where the line’s point on the x-axis of a goal represents the extent of the goal’s satisfaction)
Thus, the previous table showing link contributions is only valid if each of the goals is satisfied to the specified target level (unlikely).

Why do we need a Special Tool?

We tried to do it without a tool (instead using a CASE/diagramming tool) and quickly realised that the goals are highly interrelated, but the visual complexity of recording these links led to the omission of links. This quickly led to "silos".

This led to Requirements Engineers spending ~50% of the interaction time in the program on diagram layout (minimizing edge collisions). Since it is the underlying model that matters, and not the visual representation, we place nodes using the GraphViz library.
Evaluation & Future Work

Evaluation is constant and on-going as the methodology (and PhD) evolves – but feedback from industry so far is good.
- The 1st version implemented at Rolls-Royce resulted in:
  - New business objectives elicited (from software requirements).
  - New software requirements elicited (from business objectives).
  - Requirement priority adjusted using additional business information.
  - Reductions in non-functional requirement threshold levels.
  - Earlier identification of false assumptions about benefits to be achieved.

Future work is to improve the accuracy of goal-goal contributions by:
- using stakeholder networks (as in StakeRare) and "wisdom of the crowd" theory to capture multiple sets of contribution forecasts
- using similarity analysis on previous projects to find similar data to base estimates upon

Closing Remarks

- Challenges/difficulties:
  - There’s a good reason that contribution scores like {Low, Medium, High} are used! (normalised & doesn’t require data)

Closing Remark: The Importance of Vision

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.
"I don’t much care where--" said Alice.
"Then it doesn’t matter which way you go," said the Cat.
"--so long as I get SOMEWHERE," Alice added as an explanation.
"Oh, you’re sure to do that," said the Cat, "if you only walk long enough."

(Alice’s Adventures in Wonderland, Chapter 5)

A Brief Intro to GORE Goal Graphs

Goal graphs are directed acyclic graphs (made popular by AI), which answer “why” questions when traversed upwards, and “how” questions when traversed downwards.

The power in goal graphs is that, once complete, an agent can understand exactly what must be done in order for the abstract goal to be achieved.

Why do we need a Special Tool? (2/2)

- Additionally, existing Goal Oriented Requirements Engineering Modelling tools (such as OpenOME, Objectiver) and UML tools (such as Enterprise Architect and Magic Draw) don’t allow you to associate specific data to link elements, e.g., the contribution function.

So, we can record that x contributes to y where x is a functional requirement and y is a business objective - but the data and metadata that defines that relationship cannot be stored, e.g., the extent of the contribution or the confidence associated with that extent.

This is a problem, since the extent of a link’s contribution is key:
- For Functional requirements – a business objective’s satisfaction depends on the extent of the contribution made by software functionality.
- For Non-Functional requirements – every system is reliable, efficient, secure, etc. to some extent – it is the extent that needs to be satisfied.
What are we Trying to Relate (or Trace)?

- **Software requirements** "must say why a system is needed, based on current or foreseen conditions" as well as "what system features will serve and satisfy this context". - Ross & Schuman, 1977

- How can we demonstrate that design automation **software requirements** are well aligned to the **goals of the business** (i.e. it's chosen strategy)?

- **Business goals** must be satisfied on a continuing basis to effectively attain the Vision of the business. - OMG Business Motivation Model
Modelling the Strategic Alignment of Software Requirements using Goal Graphs

Richard Ellis-Braithwaite\textsuperscript{1} \hspace{1cm} Russell Lock\textsuperscript{1} \hspace{1cm} Ray Dawson\textsuperscript{1} \hspace{1cm} Badr Haque\textsuperscript{2}

\textsuperscript{1}Loughborough University \hspace{1cm} \textsuperscript{2}Rolls-Royce Plc.
Leicestershire, United Kingdom \hspace{1cm} Derby, United Kingdom
\{r.d.j.ellis-braithwaite@lboro.ac.uk, r.lock@lboro.ac.uk, r.j.dawson@lboro.ac.uk, badr.haque@rolls-royce.com\}

Abstract—This paper builds on existing Goal Oriented Requirements Engineering (GORE) research by presenting a methodology with a supporting tool for analysing and demonstrating the alignment between software requirements and business objectives. Current GORE methodologies can be used to relate business goals to software goals through goal abstraction in goal graphs. However, we argue that unless the extent of goal-goal contribution is quantified with verifiable metrics and confidence levels, goal graphs are not sufficient for demonstrating the strategic alignment of software requirements. We introduce our methodology using an example software project from Rolls-Royce. We conclude that our methodology can improve requirements by making the relationships to business problems explicit, thereby disambiguating a requirement’s underlying purpose and value.

Keywords—Requirements Engineering; Strategic Alignment; Quantified Goal Graphs; Requirements Traceability

I. INTRODUCTION

The stakeholders of a software project should share an understanding of the potential business benefit that a software requirement offers. If such an understanding can be achieved, the likelihood that a solution will satisfy a real business problem will be improved. Although such statements may sound obvious, it has been reported that 45% of software requirements are never deemed to be useful after implementation [1]. These unnecessary requirements cause costs and delays that perhaps could have been avoided by benefit analysis. The existence of a requirement should be questioned if it does not demonstrate potential to offer value to the business. Conversely, valuable requirements are at risk of being de-prioritised if they fail to demonstrate their potential benefit. In an organisational setting, business benefit can be gained from an alignment to business strategy.

Technically worded requirements or solution oriented requirements (i.e., specified for the machine rather than for the world [2]) hide the business problem to be solved and leave stakeholders with little understanding of the potential value. It is therefore important that the strategic alignment of such a requirement is explored in order to avoid wastage.

This paper explores the suitability of goal graphs for demonstrating a software requirement’s strategic alignment. Current Goal Oriented Requirements Engineering (GORE) standards, such as GRL [3], do not quantify the contribution one goal makes to another using metrics from the application domain, opting instead to use scales such as high, medium and low, or numerical scales such as 0-9. As a result, any strategic alignment proposed by the use of goal graphs is not specific, measurable or testable. Proposed extensions by Van Lamsweerde [4] do not consider that a chain of linked goals may contain a variety of metrics that need to be translated in order to demonstrate strategic alignment. Additionally, the current methods do not consider how the contribution score is calculated and how that affects the credibility and accuracy of the proposed benefits. This paper attempts to demonstrate how the above problems can be addressed, thereby allowing goal graphs to be used to analyse the strategic alignment of software requirements. Our methodology complements frameworks that require business value analysis, such as value-based software engineering [5], by making assumptions about business value explicit.

We have developed and implemented our methodology in partnership with an industrial partner (Rolls-Royce) to ensure its utility in real world settings. We use examples in the context of a software project to be implemented in the Transmissions Structures & Drives (TS&D) Supply Chain Unit (SCU). The software will automate geometry design and analysis for engine components, as well as for their manufacturing tools such as casting molds. Simply put, engineers will input the desired design parameters and the software will output the component’s geometry.

In Section II, we introduce the problem that this paper addresses, while in Section III, we present and evaluate the extent to which existing solutions address it. Section IV presents our methodology and tool as an extension of an existing GORE methodology in order to address the gaps outlined in Section III. We conclude in Section V with remarks on the paper’s contributions and future work.

II. THE PROBLEM

Ross and Schoman stated that software requirements “must say why a system is needed, based on current or foreseen conditions” as well as “what system features will serve and satisfy this context” [6]. Popular Requirements Engineering meta models [7], [8] and templates [9], [10] tend not to focus on “why”, typically addressing it by stipulating that rationale be attached to a requirement. However, rationale is not always an adequate description of why the requirement is valuable to the business. If only one “why” question is asked about the requirement then the rationale can still be distant from the true problem to be solved (i.e., the essence of the requirement), and it may be defined without consideration of its wider implications.

As an example of the problem that this paper examines, we introduce the following high-level requirement taken from our example software project: “While operating in an..."
analysis solution domain and when demanded, the system shall run analysis models”. The business value of this software requirement and the underlying problem to be solved is not immediately obvious, so to better understand the need for this requirement, we examine the attached rationale: “So that structural integrity analysis models can be solved as part of an automated process”. The requirement’s benefit to the business and its alignment with strategy are still not clear after one level of abstraction above the requirement. Additionally, the extent of the problem to be solved is not explained, i.e., the problems associated with solving structural integrity analysis models manually and the wider implications of doing so. Perhaps the manual process is costing the business in terms of human resource time or inaccuracy of the analysis due to error. If so, what is the business impact and how far can it be reduced? Additional broader implications may exist that are not immediately obvious - it might be the case that design innovation is constrained by the slower manual process. Clearly there is more to the rationale than is written, and arguably more than it would be sensible to express within a requirement, partly due to the duplication this would incur; several requirements may achieve the same business benefit but at varying levels of contribution and with potentially complex dynamics.

In summary, this paper argues that the strategic alignment of a requirement should be examined so that:
1. The root of the requirement can be understood so that the software can solve the right problem.
2. The extent of the problem can be understood to prove the requirement’s value and validity.
3. The value of the requirement can be understood to better inform prioritisation and project funding.

III. BACKGROUND

The following areas of research are related to the strategic alignment of software requirements: (A) Goal Oriented Requirements Engineering, (B) Strategic Alignment and (C) Software Metrics.

A. Goal Oriented Requirements Engineering

Goal Oriented Requirements Engineering (GORE) seeks to provide answers to the, so far, largely unanswered question of “why” software functionality should exist through the use of goal graphs. Van Lansweerde defines the term goal in the context of GORE as an optative statement (i.e., desired future state) about an objective that the system hopes to achieve [11]. “Goal” in the context of GORE is therefore more concerned with the goals of the system than the goals of the business. Furthermore, this definition of goal does not differentiate it from an objective. In order to relate the goals of the system to the goals of the business, we need an integrated definition of the terms used in business strategy. The Object Management Group (OMG) defines these terms in the Business Motivation Model (BMM) [12]. The BMM defines a goal as an indication of “what must be satisfied on a continuing basis to effectively attain the vision of the business”. An objective is then defined as a “statement of an attainable, time-targeted, and measurable target that the enterprise seeks to meet in order to achieve its goals”.

Objectives therefore contrast with goals in that “goals are allowed to be unrealistic and unachievable” [13]. Attempting to prove strategic alignment to non-specific goals such as “maximise profit” would be difficult since it would not be possible to prove the extent of its satisfaction. Therefore, requirements should be abstracted to objectives rather than goals for strategic alignment. Fortunately, business strategies are usually decomposed into objectives that follow SMART [14], which allows contribution between objectives to be specified, e.g., “objective x will satisfy half of objective y”.

Since the only significant difference between an objective and a goal is in its hardness and specificity (i.e., whether its satisfaction can be determined), GORE methodologies can still be applied. The most well-known GORE methodologies include KAOS [15], 9* [16] and GRL [17]. Such methodologies produce goal graphs whereby goals at a high level represent the end state that should be achieved and lower level goals represent the means to that end. The relationships between goals are typically expressed as means-ends links with AND/OR refinement. Additional elements such as agents, obstacles and dependencies are typically included. A goal graph is traversed upwards in order to understand why a goal should be satisfied and downwards to understand how that goal could be satisfied.

Three methods for applying weights to goal-goal contribution links in goals graphs were proposed by Van Lansweerde [4] with the intention of extending KAOS, but the concepts could be applied to any GORE method:
1. Subjective qualitative scores e.g., --, --, ++.
2. Subjective quantified scores e.g., 0 to 100.
3. Objective gauge variables (i.e., a quantity prescribed by a leaf soft goal to be increased, reduced, etc.).

After evaluating the above options, Van Lansweerde concluded that the specification of link weight scores with objective gauge variables is the most appropriate approach due to its verifiability. Van Lansweerde goes on to demonstrate how alternative goal (or requirement) options can be evaluated by estimating the contribution a goal makes to soft goals. Soft goals are typically qualitatively stated desires used for comparing alternatives, but in the context of [4], fit criteria quantify them. A number of translations between soft goals often need to be made in order to link requirements to business goals, which inevitably involves translating metrics (e.g., reducing component design time contributes to reducing component costs). However, the method presented in [4] does not provide guidance on propagating contribution scores to high level (i.e., abstracted) soft goals. This is important since a requirement may contribute positively to system level goals, but only slightly to higher business goals. The only propagation approach prescribed is cumulative addition (a goal’s contribution is the sum of the contributions made by the goal’s children), which cannot be applied because two scales cannot be summed.

Goal Requirements Language (GRL) was recently made an international standard through ITU specification Z.151 as part of the User Requirements Notation (URN) [3]. GRL integrates the core concepts of 9* and NFR [17]. Link contributions in GRL are specified with subjective quantified contribution scores, much the same as outlined in Van
Lansweerde’s paper [4]. For example, the time reduction goal might contribute to the cost saving goal with a contribution weight of 67 out of 100. This contribution score is untestable and meaningless; moreover it is not refutable, which, according to Jackson [2], means that the relationship is not described precisely enough because no one can dispute it. The only way such scales could be meaningful is if the goals were specified with fit criterions (e.g., a cost to be saved) and if the scales implied percentage satisfaction (which they do not). In which case, a 50/100 contribution might imply that 50% of a £20,000 annual cost saving will be achieved. However, this is only applicable for goals whose satisfaction upper bound is 100%, since the scale’s upper bound is 100%; which is not the case for goals involving increases, which may specify more than 100%. Recently, the JUCMNav tool allowed for the relation of Key Performance Indicators (KPIs) to goals in GRL [18]. KPIs specify business targets that measure the performance of a business activity. However, since KPIs do not affect the way in which contribution is measured (i.e., the contribution that a chain of goals makes to a KPI), subjectivity and ambiguity still exists.

One of the most popular tools to compare product qualities with customer requirements is the House of Quality (HoQ) diagram [19]. The fundamental failing of the HoQ is that the score values used to measure the strength of the contribution are subjective, much like those used in GRL. Additionally, since the HoQ is constructed using a 2D grid, only two dimensions can be compared in the same grid, i.e., requirements can be related to software goals, but if those software goals are to be related to customer or business goals, then additional grids will be required for each extra dimension. If these dimensions are not explored (e.g., if the software project goals are not abstracted to business goals), then the goals that the alternative solutions will be evaluated against may be incorrect (e.g., solution specific or aiming to solve the wrong problem). GORE methodologies which evaluate alternative solutions against their effect on goals, e.g., [20], also depend on the alignment of those goals to higher level goals for the resulting decision to be correct.

B. Strategic Alignment

One of the most suitable methodologies for relating software requirements to business strategy is B-SCP [21], due to its tight integration with the OMG’s Business Motivation Model (BMM). B-SCP decomposes business strategy towards organisational IT requirements through the various levels of the BMM (e.g., the vision, mission, objective, etc.). However, B-SCP cannot accurately show that a requirement satisfies a strategy since no contribution strengths are assigned to links. Since strategic alignment depends on the extent to which the strategy is satisfied (e.g., for the goal “reduce costs”, the extent is the amount of cost to be reduced), the extent to which a goal contributes towards another needs to be considered. Indeed, a large proportion of software requirements will only partially satisfy the strategic objectives. Moreover, B-SCP’s methodology refines business strategy towards IT requirements, which means that completeness of the model is dependent on the completeness of the business strategy, i.e., there is no opportunity to refine software functionality upwards to propose new business strategy. Additionally, B-SCP does not consider goal conflicts, dependencies, actors or obstacles, as in the GRL and KAOS methodologies.

The Balanced Scorecard and Strategy Maps [22] approach offers guidance on formulating and relating business goals to each other under four perspectives: financial, customer, internal processes, learning and growth. The approach does not concern software requirements; but such an approach could be performed before software requirement to business strategy alignment analysis takes place, in order to ensure business strategy completeness.

C. Metrics

Fit criterions as specified by Volere [9] and Planguage [23] can be attached to requirements in order to make them measurably satisfiable. However, assumptions made about the benefits that may be reaped after satisfaction of a fit criterion are not addressed in either methodology. Additionally, Volere and Planguage propose textual representation of requirements; and as such, relationships between requirements are hard to maintain, understand, and visualise. GQM+Strategies™ [24] was developed to extend the Goal Question Metrics methodology by providing explicit support for the relation of software metrics measurement effort (e.g., measuring the impact that pair programming has on quality) to high-level business goals. However, the approach falls short in areas similar to the other methodologies reviewed; contribution links between goals are not quantified (i.e., assumed benefit), there is a fixed number of goal abstraction levels per diagram and there are no additional concepts that are typically included in GORE methodologies to place the goals into context (e.g., actors, conflicts, AND/OR refinement).

IV. METHODOLOGY

We propose that GRL goal graphs can be used to demonstrate strategic alignment by linking requirements as tasks (where the task is to implement the requirement) and business objectives as hard goals (where the hard goal brings about some business benefit) with contribution links (where the requirement is the means to the objective’s end). The requirements should be abstracted (asking “why?”) until they link to business objectives. We have used GRL’s notation because it is part of the Z.151 international standard [3] and because its notation is well known (it originates from i*).

Soft goal elements (e.g., goals and visions from the Business Motivation Model) should not be defined in the goal graph for the purpose of demonstrating strategic alignment since their satisfaction is often immeasurable (if their satisfaction is possible at all); therefore, it is nonsensical to consider that a requirement may either partially or completely satisfy a goal or a vision. However, since objectives exist to quantify goals, and since goals exist in order to amplify the vision of the business [12], non-weighted traceability between an objective and its goals (and their related visions) should be maintained for posterity.

For our reference implementation, we have used the Volere requirements template to define the attributes of a
requirement, primarily because it specifies a fit criterion field used for testing the requirement’s satisfaction. An ‘estimated effort’ field (specified in person-hours) could be added to the template so that cost-benefit analysis can be performed. Software implementation effort estimation methods such as COCOMO [25] could be useful in refining estimated values.

We define objectives using our modified GQM+Strategies formalisation template [26], as shown in Figure 1. Our modifications to the textual template attempt to improve integration with visual GRL diagrams through:

1. The addition of the scale concept from Planguage, which specifies the metric used for measurement. An objective’s contribution to another is then given in terms of the second (parent) objective’s scale.
2. The specification of the objective’s activity attribute in the past tense, since objectives represent a desired outcome rather than an activity.
3. The removal of the constraints and relations fields since these can be expressed diagrammatically.
4. The addition of the author field so that newly proposed objectives can be identified and traced.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Reduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>TS&amp;D Fabricated Structure Manufacturing</td>
</tr>
<tr>
<td>Focus</td>
<td>Lead Time</td>
</tr>
<tr>
<td>Magnitude</td>
<td>3 months</td>
</tr>
<tr>
<td>Scale</td>
<td>Time in months required to have parts manufactured from the inception of a new engine</td>
</tr>
<tr>
<td>Timeframe</td>
<td>1 year after system deployment</td>
</tr>
<tr>
<td>Scope</td>
<td>Transmissions Structures &amp; Drives (TS&amp;D) SCU</td>
</tr>
<tr>
<td>Author</td>
<td>John Smith (Component Engineer, TS&amp;D)</td>
</tr>
</tbody>
</table>

Figure 1: Example GQM+Strategies Formalisation

An objective is satisfied when the specified magnitude is achieved within the specified timeframe. The contribution links going toward an objective specify how that magnitude will be achieved (or exceeded). If the contributions of the child objectives additively amount to meet or exceed the objective’s specified magnitude, then the satisfaction of the objective can be considered more likely than not.

Figure 2 shows the GRL notation that is used to represent the requirements and objectives. Other notations could be used on the condition that they support the same concepts.

![Figure 2: GRL Diagram Notation](image)

In order to visualise the objectives specified with the GQM+Strategies template in a goal graph, we use GRL hard goal elements with the naming syntax: “Activity[Object Focus][magnitude]”. We represent software requirements as tasks (i.e., the task of implementing the requirement) using the naming syntax: “{F/NF}[Requirement[Fit Criterion]]”, where “F/NF” is either Functional or Non-Functional, “Requirement” is a short headline version of the requirement description, and “Fit Criterion” is the short-hand version of the metric used to test the requirement’s satisfaction.

A contribution link between a requirement and an objective specifies that the satisfaction of the requirement (tested by its fit criterion) will achieve some satisfaction of the objective, where the extent of the satisfaction is defined by the contribution specified by the link. A link between two objectives is similar, except for that the satisfaction of an objective is measured by its magnitude rather than by a fit criterion. An “OR” contribution specifies that if there are multiple “OR” links, a decision has to be made about which should be satisfied. An “AND” contribution specifies that all “AND” links are required for the objective to be satisfied. A decomposition link decomposes a requirement into a more specific requirement, much like SysML’s “deriveReq” link stereotype [8]. Figure 3 shows an example diagram produced by the methodology, which demonstrates the usage of the elements in Figure 2 to explore and visualise the strategic alignment of three high-level software requirements.

![Figure 3: Example Strategic Alignment Diagram](image)

The high-level software requirement (3) in Figure 3 is decomposed to two lower level software requirements (1 & 2) to represent the hierarchy of requirement abstraction. Such refinements through decomposition links will continue until the lowest level of requirements are represented. The decomposed requirements (1 & 2) then link to objectives (4 & 5) with contribution links in order to represent what those requirements hope to achieve. The contribution links (E & F)
are of the “AND” type, since both objectives (4 & 5) are required if objective (6) is to be satisfied.

Table 1 shows a sample of the quantifications that complement the diagram. They have been separated out of the goal graph due to space constraints, but ordinarily would be annotated on the edges (connecting links) of the graph.

<table>
<thead>
<tr>
<th>Link</th>
<th>[Contribution]</th>
<th>[Activity]</th>
<th>[Scale]</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (1→4)</td>
<td>[80%] Reduction in</td>
<td>Geometry Creation Time</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D (2→5)</td>
<td>[50%] Reduction in</td>
<td>Integrity Check Time</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>E (4→6)</td>
<td>[20%] Reduction in</td>
<td>Time Required to Design</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F (5→6)</td>
<td>[13%] Reduction in</td>
<td>Time Required to Design</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>G (6→7)</td>
<td>[3 months] Reduction in</td>
<td>Manufacturing Lead Time</td>
<td>0.75</td>
<td></td>
</tr>
</tbody>
</table>

The quantified contributions in Table 1 tell us that objective (4) will be satisfied if requirement (1) is satisfied, since objective (4)’s required magnitude of satisfaction (80%) will be contributed by link (C) (80%). It is important to note that where percentages are used as contribution weights on links, this does not infer that a certain percentage of the objective’s magnitude will be achieved (in this case, 80% of 80%). Instead, the focus of the objective (e.g., geometry creation time) will be affected by that percentage in the context of the activity (e.g., a reduction by 80%). Objective (4) is then abstracted until the benefits are expressed in terms of high-level business objectives, which disambiguates estimated business value by placing the quantifications into context (i.e., a large saving from a small cost may be less than a small saving from a large cost).

Confidence levels allow users to represent how sure they are that achieving the first objective (or requirement) affects the second objective by at least the specified contribution.

<table>
<thead>
<tr>
<th>Confidence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>Poor credibility, no supporting evidence or calculations, high doubt about capability</td>
</tr>
<tr>
<td>0.5</td>
<td>Average credibility, no evidence but reliable calculations, some doubt about capability</td>
</tr>
<tr>
<td>0.75</td>
<td>Great credibility, reliable secondary sources of evidence, small doubt about capability</td>
</tr>
<tr>
<td>1</td>
<td>Perfect credibility, multiple primary sources of evidence, no doubt about capability</td>
</tr>
</tbody>
</table>

The confidence level concept is similar to that used by Gilb for impact estimation [23], so we base our confidence levels on a similar scale in Table 2. Basic confidence adjustment can be performed by multiplying contributions by their associated confidence level so that users are reminded of the impact confidence has on estimations. For example, when confidence levels are taken into consideration in Table 1, the satisfaction of requirement (1) still leads to the full satisfaction of objective (4). However, when confidence levels are considered for links (E & F), the satisfaction of objective (6) is in doubt, since \((20\times 0.8) + (13\times 0.75)\) is less than the 33% required by the objective’s magnitude attribute. Additional confidence levels can be applied to the user’s estimations to represent how qualified that user is at providing estimations. For example, someone who has implemented similar systems should be able to provide more accurate estimations than someone who has not. The accuracy of previous estimates made by that person could also be considered in order to improve the reliability of the estimations (i.e., calibration of the confidence levels).

By traversing the quantified GRL goal graphs, the business value of a requirement can be calculated by the contribution it makes to business objectives. This calculation can be automated by using a graph traversal algorithm (e.g., depth-first search) to calculate how much a given requirement contributes to a business objective. This calculation could then be used to improve the outcome of requirements prioritisation methods such as the Analytics Hierarchy Process [27], since such pairwise methods depend on the practitioners understanding of a requirement’s value.

It is important to note that software engineers and business analysts may not know the objectives (or the goals and visions, for that matter) at different levels of the business (i.e., the project, the business unit, the department, the overall business, etc.). Therefore, managers should work with stakeholders to define the business objectives before the requirements can be abstracted toward them. Indeed, it is likely that some software requirements will be abstracted toward business objectives that were not previously elicited.

Where typical goal abstraction (asking “why?”) would allow a non-specific goal such as “improve the engine”, this method requires the user to be specific in how the engine is to be improved by asking for the metric that will be affected, e.g., “component lifespan” from objective (12) in Figure 3. Users may resist quantifying benefits of requirements, especially for non-functional requirements where the subject may be intangible, however, Gilb has found that it has always been possible to do so in his experience (e.g., by polling customers to quantify customer satisfaction) and has provided guidance on doing so in [23]. Even if the magnitude cannot be elicited at first, providing a scale by which the objective’s success will be measured improves the definition of the objective by reducing ambiguity.

We suggest that this methodology should be performed after the high-level requirements have been elicited, so that resources are not wasted eliciting lower level requirements that do not align well to business strategy.

Tool support (GoalViz) has been developed (free to download at [28]) to support the methodology through:

- Input support for the requirement and objective templates with prompt question generation.
- Automatic diagram drawing to focus the user on the methodology and data rather than the graph layout.
- Automatic evaluation and summarisation of chains of links to enable efficient understanding.
- Project libraries to facilitate learning about the estimated contributions made in previous projects to improve future quantification confidence levels.
• What-if analysis allowing comparison of outcomes for different inputs where there is some uncertainty.

V. CONCLUSION AND FUTURE WORK

This paper’s unique contribution is twofold. First, we have shown how quantified goal graphs can be used to visualise the alignment of software requirements to business objectives. We have shown that in order to demonstrate strategic alignment, a chain of objectives may contain different measurement scales, and, since strategic alignment is based on estimated benefit, confidence in the estimations should be made explicit. Secondly, we have shown how goal link contribution scores can be made testable by specifying them in terms of the estimated effect they will have on the parent goal’s measurement scale. Our methodology not only facilitates disambiguation of a requirement’s business value, but more importantly, it requires that the needs of the business (i.e., business objectives) are related to requirements to ensure that the software can add value to the business.

Since the requirements are abstracted to several levels of objectives, the problem to be solved will have been defined even if the requirement was originally solution oriented.

Future work will evaluate this approach against the related work detailed in Section III within different industrial settings to examine its benefit in a range of domains. We also intend to evaluate integration with SysML [8] to improve traceability to the design that will realise the requirements.

REFERENCES
Repetition between stakeholder (user) and system requirements

Richard Ellis-Braithwaite¹ · Russell Lock¹ · Ray Dawson¹ · Tim King²

Abstract Stakeholder requirements (also known as user requirements) are defined at an early stage of a software project to describe the problem(s) to be solved. At a later stage, abstract solutions to those problems are prescribed in system requirements. The quality of these requirements has long been linked to the quality of the software system and its development or procurement process. However, little is known about the quality defect of redundancy between these two sets of requirements. Previous literature is anecdotal rather than exploratory, and so this paper empirically investigates its occurrence and consequences with a case study from a UK defense contractor. We report on a survey of sixteen consultants to understand their perception of the problem, and on an analysis of real-world software requirements documents using natural language processing techniques. We found that three quarters of the consultants had seen repetition in at least half of their projects. Additionally, we found that on average, a third of the requirement pairs’ (comprised of a system and its related stakeholder requirement) description fields were repeated such that one requirement in the pair added only trivial information. That is, solutions were described twice while their respective problems were not described, which ultimately lead to suboptimal decisions later in the development process, as well as reduced motivation to read the requirements set. Furthermore, the requirement fields considered to be secondary to the primary “description” field, such as the “rationale” or “fit criterion” fields, had considerably more repetition within UR–SysR pairs. Finally, given that the UR–SysR repetition phenomena received most of its discussion in the literature over a decade ago, it is interesting that the survey participants did not consider its occurrence to have declined since then. We provide recommendations on preventing the defect, and describe the freely available tool developed to automatically detect its occurrence and alleviate its consequences.

Keywords User requirements · Stakeholder requirements · System requirements · Duplicate detection · Redundancy

1 Introduction

Requirements documents written in natural language are usually the primary, and often sole means of communicating the desired capabilities of a software system to its developers or procurers [1]. Since requirements define the design problem to be solved [2], their quality has long been recognized as an important factor in the success of a software project. Researchers have recently investigated duplication within software requirements documents [3] in the context of minimizing the “redundancy” quality defect [4]. That is, where requirements engineers have not adhered to the “Don’t Repeat Yourself” principle: “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system” [5]. i.e.,
within a project’s requirements set. However, duplication between the different types of requirements documents—namely stakeholder (or user) and system—has yet to be investigated empirically. We propose that between-document redundancy is at least as concerning as within-document redundancy; for as well as inheriting many of the undesirable effects caused by cloning requirements or code [3, 6], engineering process-related problems may be indicated. For example, perhaps “flow down” from stakeholder needs to system capabilities could be improved [7].

Hence, the subject of this paper is a pattern of repetition that occurs between a project’s requirements documents while following the generic “V” systems engineering model [8], i.e., where stakeholder requirements (problems) are transformed into system requirements (abstract solutions). In confusion over a plethora of conflicting requirements engineering (RE) terminology and guidelines [9], practitioners are sometimes led to write (or rather re-write) system requirements (SysR) that add insignificant information to their associated user requirements (UR), or vice versa. For example, the UR “the user shall be able to write an email”, and the SysR “the system shall enable the user to write an email”. From an information-theoretic viewpoint [10], these requirements could be considered duplicates, since the difference between them expresses information known about the whole set of SysRs. Such duplication renders traceability between the two statements an inefficient use of the reader’s time, where it should have provided a source of rationale [11] and translation between “the voice of the customer” and “the voice of the engineer” [12]. Nevertheless, both documents (UR and SysR) must still be read by architects and developers to ensure pertinence, since it is their responsibility to “find out the real story behind the requirements” [13]. While doing so, reading the occasional repeated requirement statement may evoke feelings of déjà vu, but it is more concerning when a significant proportion are repeated, since:

1. Duplication between URs and SysRs indicates a misunderstanding of their role, and therefore potential omission of either the problem or solution description, indicating concerns about the SysR’s “flow down”;  
2. Finding worthwhile non-duplicated requirements becomes akin to “finding a needle in the haystack”, hence risking ignorance of important requirements (caused by readers losing the will to continue reading).

More than a decade ago, Kovitz [14] and Alexander [15] independently identified and discouraged the reportedly common practice of repeating the description between user requirements and their associated system requirements. Since this paper’s authors still witness this practice in industry, the following research questions (RQs) were constructed:

- **RQ1** According to the literature, what are the differences supposed to be between a UR and a SysR?  
- **RQ2** Does repetition between UR and SysR statements (still) occur, and if so, how often?  
- **RQ3** Why does UR–SysR repetition occur, and what does it indicate about the project?  
- **RQ4** Is UR–SysR repetition a problem, and if so, why?  
- **RQ5** Can the problems caused by UR–SysR repetition be reduced in existing and future requirements sets?

In summary, this paper sets out to explore repetition between pairs of UR and SysR statements and to discuss whether it can be considered a problem in light of the pertinent RE literature. To provide more than anecdotal evidence in answering RQs 2–5, we present the results of a case study from a UK defense consultancy. We discuss a survey of sixteen software systems engineering consultants, and an analysis of UR–SysR requirement pairs from two independent software projects. While doing so, we describe the requirements traceability analysis tool (free to download) [16] developed:

- to enable the investigation of RQs 2 and 5;  
- to support both producers and consumers of UR and SysR documentation in future projects;  
- under the wider research aim of increasing the usability of requirements quality assessment.

## 2 Background and terminology

The following sections will introduce the concepts of requirements at the user, system, and software level (2.1 and 2.2), traceability between them (2.3), and similarity analysis (2.4).

### 2.1 Requirements

There is little consensus on the precise meaning of the terminology used in RE—especially on the term “requirement”. Jureta et al. [9] recently summarized that it is “variously understood as (describing) a purpose, a need, a goal, a functionality, a constraint, a quality, a behavior, a service, a condition, or a capability”. Fifteen years before that, Harwell et al. [17] asserted that “if it mandates that something must be accomplished, transformed, produced, or provided, it is a requirement—period”. What both of these definitions share in common is that a requirement is either about a problem (i.e., a purpose, need, or goal to be accomplished) or about a solution (i.e., a functionality, quality, behavior, service, or capability to be transformed, produced, or provided). As Wieringa puts it, there are “two schools of thought” in RE [18], and confusion among practitioners about which “school” their requirements should adhere to [15] seems inevitable due to the semantic
overload of the “requirement” term. Idealistic and ambiguous guidelines, such as that requirements should describe what software must do rather than how it will do it, only complicate the issue [19], since as Kovitz [14] eloquently puts it, “everything that a piece of software does is what it does, and everything that a piece of software does is how it does something”.

2.2 Stakeholder, system and software requirements

Qualifying adjectives, such as those in this section’s heading, are often prepended to “requirement” to indicate which “school of thought” the requirement belongs. Popular requirements [11], systems [2, 20], and software [21] engineering textbooks, along with international systems and software requirements engineering standards [22–24] primarily refer to two such qualifiers: stakeholder and system. In Table 1, we distill the definitions for these types in terms of their domain (problem or solution?), role (for what purpose?), and language (whose vocabulary is it written in?), according to these sources. First, however, we describe the scope of these qualified requirement types in terms of their subtypes.

2.2.1 Stakeholder requirements subtypes

Given that a “stakeholder” is “an individual, group of people, organisation, or other entity that has a direct or indirect interest in a system” [11], then business, legal, or user requirements could be considered as subtypes of stakeholder requirement. For example, “business requirements” describe the “needs of the enterprise”, rather than those of “a particular stakeholder or class of stakeholders” [22]. Confusingly, this terminology is occasionally used interchangeably. Throughout this paper, we continue the convention (considered to be “incorrect” [25, 26]) of referring to “stakeholder” requirements as “user” requirements, for coherence with the previous works on the topic [14, 15].

2.2.2 System requirements subtypes

Since a system is “a collection of components—machine, software, and human, which co-operate in an organised way to achieve some desired result” [11], there can be requirements on the whole system, its components, and on its interfaces. Hence, a software requirement is a system requirement by definition, but a system requirement is not necessarily a software requirement, since software requirements are enforced solely by the software agent to-be, and “a system does not merely consist of software” [25]. Whether or not a system requirements document contains non-software requirements (such as expectations on human behavior, hardware, or interfaces), depends on whether the software will provide “essentially all the functionality” of the envisioned system (as in “Software-Intensive Systems” [20]), or whether it will be just one part of a larger system [22].

2.2.3 Difference between stakeholder and system requirements

To summarize Table 1 and to answer RQ1, translating a UR to a SysR should involve at least:

1. A decision on the abstract system-to-be. Not doing so risks that the SysRs will be too ambiguous to be useful. For example, deriving specific SysRs on energy usage that would be equally applicable to burglar alarms, electric fences, and guard dogs is a difficult task [27].

2. A transition from a stakeholder need to a system responsibility. Not doing so risks that the need will remain unsatisfied, and indicates that a desire may not have been clarified, decomposed, derived, or allocated [28], such that it is clear that a component (or more generally, the system) will be causally responsible for it.

3. Technical, rather than customer-oriented language. Not doing so risks that software developers misinterpret requirements and waste effort on software that the customer will not accept; Rework in programming costs exponentially more time than in requirements [29].

The unanimous recommendation from Table 1 that URs should describe problems while SysRs should describe solutions cannot be interpreted as an absolute rule, since problems and solutions are not conceptually orthogonal. For example, Jackson notes that merely structuring a problem moves the description toward a solution [36], while Berry concludes that specifying a problem precisely sometimes requires reference to the solution domain, and so “for some requirements, it may be impossible to specify what without saying something about how” [37]. Furthermore, a UR or a SysR can simultaneously be a description of a problem and a solution, since what is an end in one context is a means to an end in another (as is the pragmatist’s argument against “intrinsic value” [38]). Similarly, Zave and Jackson argue that “almost every goal is a subgoal with some higher purpose” [39]; adapted to this UR–SysR context is that even the most non-technical of URs will rarely not be “solutions”, e.g., to achieving stakeholder happiness. (Consequently, alternative solutions to the “engineering problem” should be scoped to the SysR level to avoid requirements specifying that the stakeholder should instead “adopt religion or devotion to family” [39], for example.) Overall, the essence of the matter is not that there should be a problem–solution dichotomy, but rather...
<table>
<thead>
<tr>
<th>Domain</th>
<th>Role</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Textbooks</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems Engineering (Stevens et al.)</td>
<td>“<strong>problem domain</strong>” [20]</td>
<td>“<strong>defines the results</strong> that the system will supply to users” [20]</td>
</tr>
<tr>
<td>Software Engineering (Sommerville)</td>
<td>“a company… define[s],… needs” &amp; “a solution is not pre-defined” [21]</td>
<td>“so that several contractors can bid…<strong>offering different ways of meeting the…organization’s needs</strong>” [21]</td>
</tr>
<tr>
<td>Standards &amp; Frameworks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISO 29148</td>
<td>“provide the true picture of the <strong>problem</strong> to be solved” [22]</td>
<td>“<strong>describe[s] the needs</strong> that a given stakeholder has and how that stakeholder will interact with a solution”—“a bridge between business requirements and the…solution requirements” [22]</td>
</tr>
<tr>
<td>SEBOK</td>
<td>“characteristics, … and functional … requirements for a product solution” [22]</td>
<td>“provide[s] a description of what the <strong>system should do</strong> in terms of the system’s interactions or interfaces with its external environment” [22]</td>
</tr>
<tr>
<td>UK MoD AOF</td>
<td>“views of stakeholders as they relate to the <strong>problem</strong> (or opportunity)” [23]</td>
<td>“enables the characterization of the solution alternatives” [23] &amp; “the <strong>basis</strong> of SysR activities, <strong>system validation</strong> and stakeholder acceptance” [23]</td>
</tr>
<tr>
<td></td>
<td>“<strong>define[s],… the outcome … that the user needs to be able to achieve</strong>” [33]</td>
<td>“defines what is needed…” [32] in terms of “<strong>what the system must do</strong> (and) how well it must perform” [34]</td>
</tr>
</tbody>
</table>

*Table 1* Differences between user requirements (UR) and system requirements (SysR) definitions.
that both the positive and negative effects (and hence the pertinence and adequacy) of a SysR can be understood when it is related to one or many URs that describe the problem(s) that should be treated.

At a more philosophical level, in a discussion over what a UR or a SysR should be, it is crucial to remember that requirements in themselves are intangible and exist independently of their representations [40]. Indeed, aside from UR and SysR documents, requirements and their pertinence could be represented by other means, e.g., with goal graphs, as in GRL [41] or KAOS [42] (where system requirements “implement” stakeholder goals [43]). In RE, the important concern should not be the choice of representation or associated nomenclature, but rather that the “requirements problem” is solved [9], or put bluntly, that the “how”s satisfy the “what”s such that asking “why” of the “how” leads to the “what”. Then, the wider concern should be that the “what”s will be valuable, as is the topic of value-based requirements engineering [44].

Finally, a discussion on “requirement” qualifications and representations would not be complete without addressing the distinction between the so-called Functional Requirements (FRs) that prescribe desired behaviors, and Non-Functional Requirements (NFRs) that prescribe desired qualities [45]. Regardless of the myriad of definitions proffered for these, Glinz demonstrates that the distinction lies in the requirement’s representation rather than its concern [46]. For example, the different representations “…require username and password…” (FR) and “…probability of unauthorized access…” (NFR) refer to the same security concern, but provide different information about it. Similarly, a UR and a SysR may both refer to the same concern, yet their different representations should provide different information. Stated in terms of the KAOS meta-model for requirements engineering [42], if a requirement’s SysR representation is missing, then information about responsibility may be lost (i.e., which system component will enforce the requirement?), while if the UR representation is missing, then information about which stakeholder(s) wish for the requirement, and why they wish for it, may be lost.

To add illustration to the so-far conceptual discussion, Table 2 provides examples of UR–SysR pairs from the literature. Examples #1–6 are intended by the respective authors to be examples of good practice, while #7 is Kovitz’s example of the UR–SysR repetition bad practice. The reader will observe that these translations from UR to SysR vary in their adherence to the conceptual definitions in Table 1, but as Rost describes, compliance with RE standards by industry and even “semiofficial” authors is generally poor [47].

### Table 2 Examples of user requirement–system requirement (UR–SysR) pairs from the literature

<table>
<thead>
<tr>
<th>#</th>
<th>UR</th>
<th>SysR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“The driver shall be able to deploy the vehicle over terrain type 4A” [11]</td>
<td>“The vehicle shall transmit power to all wheels” [11]</td>
</tr>
<tr>
<td>2</td>
<td>“The &lt;Management User&gt; requires data to be protected &lt;Measure of Effectiveness&gt; from unauthorized access” [48]</td>
<td>“The &lt;Security System&gt; shall provide encryption to electronic data” [48]</td>
</tr>
<tr>
<td>3</td>
<td>“The maximum duration of a trip from the mother craft to the beach is 1 h, because trips over an hour will make most people too seasick to function effectively” [28]</td>
<td>“The landing craft shall have a minimum speed of 25 knots” [28]</td>
</tr>
<tr>
<td>4</td>
<td>“The elevator shall receive calls for up and down service from all floors of the building” [2]</td>
<td>“The elevator system shall produce digitized passenger requests” [2]</td>
</tr>
<tr>
<td>5</td>
<td>“Patients shall be medically aided within less than eight minutes on average” [20]</td>
<td>[The system shall] “Locate available ambulance” [20]</td>
</tr>
<tr>
<td>6</td>
<td>“The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month” [21]</td>
<td>“The system shall automatically generate the report for printing after 17.30 on the last working day of the month” [21]</td>
</tr>
<tr>
<td>7</td>
<td>“User shall be able to store grocery inventory data” [14]</td>
<td>“System shall store grocery inventory data” [14]</td>
</tr>
</tbody>
</table>

2.3 Traceability between user and software requirements

All of the literature cited in Table 1 recommends that relationships between SysRs and URs should be captured and maintained, e.g., with a traceability matrix such as MODAF view SV-5 [49], the House of Quality [12], or more simply as a traceability field in the spreadsheet, document, or database. The analysis enabled by such traceability can be classified as either [11]:

- **Impact analysis**—“What if this requirement is changed?” (for change analysis);
- **Derivation analysis**—“Why is this requirement here?” (for goal and cost/benefit analysis);
- **Coverage analysis**—“Are all customer concerns covered?” (for progress and accountability analysis).
The results of traceability analysis can be distilled by traceability metrics to improve the identification of risks and flaws early in the system life cycle [50]. For example, coverage metrics could be calculated to describe the percentage of URs that relate to at least one SysR. Here, complete coverage is highly desirable for convincing stakeholders that their concerns will be addressed by the system-to-be. However, the descriptive power of the coverage metric depends on the URs and the SysRs according to Table 1’s guidelines. For example, a 100 % UR coverage metric would not ensure what the metric is supposed to ensure if it is calculated from a set of repeated UR–SysR pairs. Indeed, many projects have failed to derive benefit from traceability [51, 52], e.g., because of “traceability for its own sake” or for process compliance. Hence, Asuncion et al. propose that each traceability link made throughout the system’s life should be questioned for its ability to be beneficial [51], and in the case of tracing between a repeated UR and a SysR, i.e., to form a UR–SysR pair, we do not see much benefit. In summary, and to contribute to answering RQ4, repetition between URs and SysRs diminishes the efficacy of the aforementioned traceability analysis, the impact of which ranges from misinforming stakeholders about problem–solution coverage, through to being unable to make value-oriented decisions in the software project, e.g., in prioritization, trade-off analysis, etc. [53].

2.4 Measuring the repetition in a UR–SysR pair
(string similarity from an information retrieval perspective)

Being able to objectively quantify the degree of repetition between UR–SysR pairs is essential for understanding the extent of its existence, and therefore for the management and reduction of it. Since URs and SysRs are typically represented by textual statements, even if they were derived from models such as in [54], their similarity—and hence the degree of repetition between them—can be quantified using string similarity metrics, of which, Cosine, Jaccard, and Dice are the most popular [10]. These metrics take as input two strings and output a normalized score in the interval [0,1] for comparability of different string lengths, starting at zero when there is no commonality (no matter how different), and reaching one when there is no difference [10]. These metrics are commonly calculated (e.g., in e-mail spam detection [55]) according to the “bag of words” model where a requirement’s description would be processed into a set of unordered terms (words) and their frequencies. The steps in processing a sentence to a bag of words typically include tokenization (splitting the sentence into words), and stemming (stripping words to their “base meaning”, e.g., “maintainable” becomes “maintain” [56]. The contrived problem that “user uses system” and “system uses user” would be considered equal, illustrates the drawback. However, this is not likely to be an issue, since two requirement descriptions would seldom use exactly the same frequencies of terms sequenced differently to describe different information. Besides, disregarding a term’s context (its surrounding terms) improves recall (i.e., more similarity is detected), since a term such as “performance”, will be more common between a UR–SysR pair than sequences of terms, such as “system performance”, as n-grams [56] would represent.

While the Jaccard and Dice metrics are intuitive (in essence, they divide the number of common terms by the total number of terms), their disadvantages for this paper’s context are:

1. That term occurrence is Boolean. For example, the second occurrence of the “user” term is ignored in “the user shall be able to communicate with another user”;
2. That the importance of term difference is assumed to be equal across all terms, but existence of the term “system” in a SysR is more trivial than the term “encryption”.

Therefore, modern information retrieval applications such as search engines favor a vector space model of similarity using terms as dimensions, strings as vectors, TF–IDF distance (term frequency–inverse document frequency) as vector magnitudes, and the calculated cosine measure of angular similarity between vectors [56] (as visualized by the two-term example in Fig. 1). Hence, the first aforementioned problem of the Jaccard and Dice metrics (Boolean representation of term occurrence) is treated by considering the frequency of terms in each of the strings.

The second problem of the Jaccard and Dice metrics (assuming equal term importance) is treated by TF–IDF by

![Fig. 1 Similarity between String A and String B using the cosine measure and the vector space model, as used to calculate TF–IDF (figure modified and adapted from [57], and simplified to include only two dimensions and hence only two terms)](image_url)
multiplying the term frequencies by the term’s inverse document frequency (the log-dampened probability that the term could be picked at random from the projects set of requirements). Hence, any non-similarity between a UR–SysR pair comprised of terms frequently occurring in the requirement set (e.g., the term “user” in a UR) is treated as less significant than non-similarity comprised of rarer terms. This weighting seems logical in the context of UR–SysR repetition. For example, the differences in UR–SysR pair #7 in Table 2 (-user, +system, -be, -able, -to) are likely to be common across the project’s set of URs and SysRs, and they do not seem important enough to warrant reading the SysR if the UR has been read.

The usefulness of IDF weighting is underpinned by the assumption that important terms are rare while trivial terms are common. However, take for example the term “notwithstanding”, which despite being generally rare, is not a significant indicator of a requirement’s concern. Furthermore, specially rare terms, i.e., rare in a particular requirement set but generally common, would also be given an inflated importance by IDF. An example from this paper’s case study is the replacement of the term “which” with “that” from a UR to a SysR. Hence, the impact of trivial but rare differences between UR–SysR pairs needs to be reduced by additional means. To do so, many information retrieval systems remove “stop-words” prior to calculating TF–IDF. That is, commonly occurring words such as the, that, or with are removed after tokenization, since while they “help build ideas”, they “do not carry any significance themselves” [58]. The list of 33 stop-words used in Apache’s Lucence (popular information retrieval software) provides a well-tested set [59]. Runeson et al. [57] also suggest that if the strings are structured (i.e., based on a template), then the template’s structural words should also be removed to reduce their impact on the similarity measure. The EARS templates [60] provide such a list of candidate stop-words commonly used to structure natural language requirements. However, logical operators and other high-information-value but common words exist in both Lucene’s stop-word set and the EARS template. These terms should not be removed in this paper’s RE context because they are able to radically change a requirement’s meaning with one term frequency change. Hence, Lucene’s stop-word set and the EARS keywords comprise the stop-word set for this paper, except for the {and, or, no, not, when, while, where, if, then} terms.

Finally, it is worth citing Falessi et al.’s [61] recent classification and comparison of natural language processing (NLP) techniques for identifying equivalent requirements. Falessi et al. conclude that a technique comprised of the vector space model and the cosine similarity metric, “if adopted in a traceability tool, is expected to provide the maximum benefit to the human analyst compared to other NLP techniques”. Interestingly, they observed that the use of synonyms (i.e., considering synonymous terms such as “monitor” and “screen” to be the same dimension in the vector space model) was “deleterious”, “introducing more noise [i.e., false positives] rather than making a positive contribution”. In addition, Runeson et al. [57] remark on the difficulty of constructing a domain-specific synonym list (for detecting duplicate software defect reports) that would be “efficient enough” to justify the effort. Indeed, they report an insignificant improvement after attempting to do so. Overall, Falessi et al. [61] find that “the simplest NLP techniques are usually adopted to retrieve duplicates”.

(The reader interested in learning more about TF–IDF is directed to a one-page online tutorial [62], or to Manning and Schutze’s [56] book for a more thorough treatment.)

3 Research method

This paper’s subject became of interest to the authors during the first author’s placement at a UK defense contractor as part of a wider research project [63]. We constructed the research questions introduced in Sect. 1 after three interviews with experienced consultants from the organization to ensure their relevance (mean 21.6 years experience; SD 2.8). Since the studied phenomenon is sociotechnical and cannot be accurately replicated in laboratory conditions, it must be studied in its natural context. Therefore, the case study research design is adopted, and is guided by Runeson and Höst’s [64] guidelines for case studies in software engineering. As such, our intention is not to answer the research questions for the general population of software engineering practice, since the size and variety of samples required to do so were unaffordable within the resource constraints of this project. Instead, we intend to contribute to (or in this case, start) the “chain of evidence”, and to “enable analytical generalisation where the results are extended to cases which have common characteristics” [64] (rather than generalize purely with inferential statistics [65] and its commonly “unrealistic” random sampling assumption [66]). Consequently, the sampling method used within this case study is a form of non-probability “convenience sampling”, guided by aspects of “judgement sampling”, “snowball sampling”, and “quota sampling” [67].

As further motivation for the case study research design, Easterbrook [68] argues that it is well suited for “how and why questions” due to its depth (rather than breadth) of enquiry. Finally, falsification is afforded by single case studies, and is sufficient for providing answers to our research questions on the phenomena’s incidence and consequences, especially RQ2’s “does it occur?” and
RQ4’s “is it a problem?” (i.e., only one non-pink flamingo is required to refute that “all flamingos are pink”). Lastly, since robust case studies require “triangulation” of conclusions using different types of data collection methods [64], Sect. 3.1 describes the questionnaire we employed (subjective data), while Sect. 3.2 describes the similarity analysis of UR–SysR pairs performed by our software tool (objective data). This mixed-methods approach is especially important since a practitioner’s understanding of UR–SysR repetition (including its frequency of occurrence) will likely be distorted due to cognitive biases, such as confirmation bias or the availability heuristic [69].

3.1 Research method 1: Questionnaire

In order to answer RQs 2–4 (does UR–SysR repetition occur, why does it occur, and is it a problem?), we selected consultants from the organization who were experienced in requirements engineering (n = 23) and invited them to complete an online structured questionnaire. The questionnaire was constructed with compliance to Umbach’s best practices in order to minimize potential for survey error [70]. We received completions from 16 participants (70 % response rate) reporting on 260 years of combined software engineering experience (mean 16.25, SD 5.32). The responses describe a combined total of 235 software development projects (mean 14.68, SD 8.26) in the last 10 years. While these projects are not likely to be wholly distinct, they are unlikely to be entirely homogenous due to the nature of the consulting business. We asked the participants to limit their responses to the last 10 years of their career in order to ensure the currency and relevance of the results. For each question in the survey, a “do not know or N/A” option was provided to prevent any uninformed responses distorting those provided with higher confidence. Where percentages are used to describe results in this paper, they are always relative to the total number of participants (16) rather than the number of non-“do not know or N/A” responses.

3.2 Research method 2: UR–SysR similarity analysis

In order to improve the robustness of our answer to RQs 2 and 5 (does it occur, and can problems be reduced?), we used our software tool [16] to analyze pairs of requirements documents from two software projects written by different authors, i.e., one author per project and four requirements documents in total. We were limited on the number of projects whose documentation we could acquire due to security restrictions, but it should be noted that this sample size is strong compared to other research on requirements documents from industry “due to nondisclosure agreements between researchers and industry”, or lack thereof [61]. Crucially, in order to construct a “typical” case study (of Gerring’s 9 case types [71]), we sought UR and SysR documentation that was representative of common practice, rather than of UR–SysR repetition. We were assured by the three interviewed consultants of the documents’ typicality in the organization, as well as in the UK defense IT industry—afforded by the consultants’ collaboration with other consultancies. (Confidence in the latter is obviously lower than in the former due to the local and isolated nature of a case study.)

We used our software tool to perform the following processes:

1. Import requirements from documents, each comprised of a global and a local (hierarchical) ID field, a description text field (“system/user shall…”), a fit criterion text field (test case), and a rationale text field (justification).
2. Reconstruct UR–SysR traceability links using textual references from SysRs to the global IDs of URs (thus forming the set of the project’s UR–SysR pairs).
3. Preprocess the text fields in the requirements according to Sect. 2.4, as exemplified in Table 3.
4. Compute IDF's for the project's set of URs and SysRs (where the IDF’s “document” set is comprised of only the requirement field being analyzed, e.g., the fit criterion).
5. Calculate TF–IDF distance scores for each of the project’s UR–SysR links (distance score is 1-similarity score).
6. Display the UR–SysR pairs and their TF–IDF scores for identification of false positives, as shown in Fig. 2.
7. Compute statistics (TF–IDF histograms and correlations) to summarize the detected UR–SysR repetition.

Steps 1–2 required manual conversion of the MS Word documents containing the requirement statements into CSV files via MS Excel using copy and paste. Since our tool can interactively map CSV fields to the core requirement fields [72], a pre-defined requirement field structure is not required. (This mapping and import stage would not be required if the requirements were available in the requirements interchange format ReqIF [73], e.g., when using SysML models.) Steps 3–5 were implemented using the Java LingPipe linguistic analysis library [74] to ensure robust and repeatable results. The following of LingPipe’s tokenizer factories were used: IndoEuropean, LowerCase, PorterStemmer, and Stop (with the stop-word list described in Sect. 2.4). Punctuation was removed using the regex “\p{P}”. Step 6 is assisted by our tool’s use the Java diff-match-patch library [75] and Graphviz [76] to visualize differences between a UR’s and a SysR’s tokens, and Step 7 is assisted by our tool’s use of the R statistics software [77].
Table 4 describes the number of requirements in the two projects, their traceability, and their wordiness. In Project A, all but three SysRs link to one UR (there are 55 traceability links from SysRs to URs, and 52 SysRs). This indicates that the majority of repetition discovered between UR–SysR pairs in Project A can cause the repeated requirement (either the UR or the SysR) to be redundant, since it is not linked to other requirements. However, the cost of redundancy (i.e., the time to read or write) in Project B will likely be higher than it would be in Project A, since the requirement statements contain more words. All links between URs and SysRs are of positive rather than negative polarity, and so each traceability link represents a UR–SysR pair in the sense that the SysR is some part of the solution to the UR, as opposed to conflicting with it.

From the relatively low number of requirements in Project A and B, the reader might be wondering why the degree of UR–SysR repetition is investigated using TF-IDF calculated by the software tool, instead of just manual inspection by humans. Indeed, a viable strategy might be to ask people to rate the degree of repetition and then calculate inter-rater reliability. However, one of the aims of this research is to provide automated tool support to assist quality analysis of requirements documents. The aforementioned manual inspection’s time consumption would be measured in hours (as elaborated in Sect. 4.3), while automated analysis’ time would be measured in seconds or minutes. Furthermore, manual verification by two IT professionals external to this research, found that at the chosen TF-IDF cutoff point (0.3) for classifying a UR–SysR pair as repeated, there were no false positives (i.e., UR–SysR pairs that were not considered repeated but were classified as such), and so the TF-IDF derived results provide a confident lower bound on the proportion of repeated UR–SysR pairs in this case study. (This topic, and the lesser effects of false negatives are later discussed in Sect. 4.4.1.)

3.3 Context of the case study

An overview of the projects that the participants described in their questionnaire responses is provided in Tables 5, 6, 7, 8 and 9, where the percentages describe the distribution of the participants’ responses to the questions.
The software projects’ benefits and usage seem in-line, if not better than the averages reported in a recent survey of the IT industry [78]. Interestingly, difficulty in contacting end-users (Table 7) indicates that applying agile RE approaches (i.e., no UR–SysR documentation) in this organization’s context could be challenging [79]. Furthermore, since requirements were rarely explicitly traced to business objectives (Table 9), it is especially important that the URs describe the SysRs’ corresponding problem(s) in order to contextualize and assure their pertinence and adequacy.

The majority (75 %) of the participants had experience in mostly military projects, while 18.75 % had experience in mostly civil projects, and the remainder had approximately equally split experience. As such, most of the requirements engineering practice reported on in this case study will have adhered to the guidelines in the UK Ministry of Defence’s Acquisition Operating Framework (AOF) [34]. This does not significantly limit the generalizability of the results, since as Table 1 shows, the AOF is mostly in agreement with the other standards and guidelines. Furthermore, the AOF seems opposed to redundancy in requirements documents, since it offers two redundancy-related guidelines for creating user requirements documents in [33]:

- “Exploit the hierarchy to minimize repetition. Statements can inherit characteristics from parent nodes.”
- “Avoid duplication. If two users/stakeholders have a common interest, record both sponsors against a single statement.”

These recommendations do not explicitly address redundancy between UR and SysR documents, e.g., for the first guideline, the hierarchy of URs may be decomposed differently than the SysRs, since at the UR stage, the system is not yet architected, i.e., split into functional components. However, when combined with the AOF’s conceptual differences shown by Table 1 and the exemplary differences shown by requirement #2 in Table 2, we conclude that, in theory, URs translated to SysRs in accordance with the AOF should not contain a high degree of repetition.

4 Results

This section revisits each research question defined in Sect. 1 and provides our answers to them. RQ1 was previously answered by Sect. 2.2.3, and so this section starts from RQ2.

4.1 RQ2: Does UR–SysR repetition (still) occur?

There are two aspects to this question’s answer: the proportion of projects that exhibited UR–SysR repetition (1) and the proportion of UR–SysR pairs that exhibited repetition within projects (2).

4.1.1 How many projects exhibited UR–SysR repetition?

The questionnaire asked the participants to describe how many software projects they had seen in the last 10 years that had exhibited repetition between the project’s UR and SysR descriptions. To reduce the likelihood of misinterpretation, we included Kovitz’s example of UR–SysR repetition (requirement #7 in Table 2) in the question. The participants were given five ordinal categories to choose from (chosen in place of interval or point estimates to minimize the survey dropout rate), spanning from “nearly none” to “nearly all” of their projects. Table 10 provides the frequencies of the participants’ responses to these categories. In summary, 75 % of the respondents had seen...
UR–SysR repetition in at least half of the projects they had seen in the last 10 years, and all of the respondents had seen at least some projects exhibiting UR–SysR repetition.

There was a difference in the respondents’ experience (number of projects in the last 10 years) between those who chose {~ None, Some, ~ Half} (mean 10.7 projects; SD 6.1) and those who chose {Most, ~ All} in Table 10 (mean 17.7 projects; SD 8.7). Hence, it could be hypothesized that those who saw repetition in half or less of their projects would have seen more repetition if they had been exposed to more projects. However, both the high variation (SD 8.7) of the {Most, ~ All} group, as well as the Wilcoxon rank-sum test result of $W = 16$, $p = 0.1$ (not requiring data to be normally distributed or on an interval scale [80]), do not provide support for this hypothesis, indicating no significant difference between stakeholder experience and frequency of observed UR–SysR repetition.

The questionnaire asked the participants if they had observed a higher or lower degree of UR–SysR repetition throughout their whole career, as opposed to the last 10 years. The majority of the respondents (87.5 %) selected “no difference”, while the remainder (12.5 %) selected “less repetition”, and none selected “more repetition”. In other words, according to the memory of the respondents, the incidence of UR–SysR repetition has not reduced in the last 10 years, compared to their whole careers.

### 4.1.2 How many UR–SysR pairs exhibited UR–SysR repetition?

All of the literature cited in Table 1 states that URs and SysRs are primarily comprised of three core fields: the requirement’s main description, a fit criterion (test case), and a rationale (justification). Hence, this section presents the analysis of repetition (i.e., TF–IDF scores for each UR–SysR pair) in our sample project’s requirements’ description, fit criterion, and rationale fields.

Figure 3 shows a histogram of the TF–IDF distance scores for the two projects’ UR–SysR pairs’ description fields, where low distance scores represent a high degree of repetition. It is apparent that while Project A has a significant number of almost identical UR–SysR pairs (31 %), the distribution is fairly symmetrical around the mean (mean 0.49; SD 0.40; median 0.56; skewness = −0.02). Hence, there is also a great deal of significantly different pairs that are at risk of being overlooked by the reader.

**Table 10** Projects where requirement descriptions were repeated between URs and SysRs in the last 10 years (2003–2013)

<table>
<thead>
<tr>
<th></th>
<th>~ None</th>
<th>Some</th>
<th>~ Half</th>
<th>Most</th>
<th>~ All</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 %</td>
<td>25 %</td>
<td>18.75 %</td>
<td>50 %</td>
<td>6.25 %</td>
<td></td>
</tr>
</tbody>
</table>

Project B appears to exhibit less UR–SysR repetition since it is skewed toward a higher degree of difference (mean 0.65; SD 0.27; median 0.75; skewness = −1.02). In summary, 43 % (24) of Project A’s and 16 % (16) of Project B’s UR–SysR pairs’ description fields have a TF–IDF distance score of up to 0.3. (The 0.3 threshold for classifying a repeated UR–SysR pair was derived from manually assessing the triviality of the differences between the UR–SysR pairs, as is later discussed in Sect. 4.4.1.) Hence, on average over the projects, nearly a third of the UR–SysR pairs’ description fields were repeated with no significant textual differences.

To demonstrate the meaning of the TF–IDF distance scores shown in Fig. 3, examples of UR–SysR description fields from both projects are provided in Table 11. The requirement descriptions are presented as the output of Myer’s difference algorithm with semantic cleanup applied [75] in order to improve the reader’s comprehension of the differences. The visual representation of the differences allows the reader to quickly see that most of the listed UR–SysR pairs are not parsimonious (i.e., where the less data required to convey the message, the better). This edit-based model of string similarity can be used to measure the degree of similarity by counting the number of insertions and deletions required to transform one string to the other, e.g., the Levenshtein distance [81], as is used in the diff-match-patch library [75].

However, edit-based similarity metrics are highly sensitive to changes in the sequence of words, and do not attempt to distinguish between important and trivial differences. In this case study, we observed that the Levenshtein and TF–IDF distances for the description fields were more strongly correlated in Project B (Pearson’s coefficient = 0.88) than Project A (Pearson’s coefficient = 0.80). Hence, more of Project A’s UR–SysR differences were either common, rare, grammatical, or word-order related. This information could be useful to prioritize the review of UR–SysR pairs, since disagreement between a Levenshtein and a TF–IDF score indicates that the pair may seem more similar or different to the reader.
than they actually are. Finally, Table 11 indicates that the scope for improving the projects’ requirements likely lies in the problem domain. For example, the URs mostly refer to “user” interactions with the solution rather than problems that a particular stakeholder (individual, class, or role) wishes to be solved by the system-to-be.

Interestingly, the rationale field was repeated far more than the description field in both projects (approximately twice as many repeated UR–SysR pairs); 80% (44) of Project A’s and 35% (34) of Project B’s UR–SysR pairs have a TF–IDF distance score of up to 0.3. Furthermore, Fig. 4 shows that a majority of UR–SysR pairs were of the “copy and paste” type (i.e., exact duplication). In fact, all of Project A’s rationale fields could be considered duplicated, since its non-similarities do not add valuable information, e.g., SysRs containing phrases such as: “assumed from user requirement”. (This type of phrase could be added to a stop-phrase list so that they are not considered to be worth reading.) On average over the projects, two-thirds of the UR–SysR pairs’ rationale fields were repeated with no significant differences.

Figure 5 shows that on average, the fit criterion field contained the most repetition between UR–SysR pairs; 80% (44) of Project A’s and 73% (72) of Project B’s UR–SysR pairs have a TF–IDF distance score of up to 0.3. As with the rationale field, most repetition tended to be almost exact duplication. The similar but not duplicated UR–SysR pairs (i.e., the second and third buckets in Fig. 5) were also insignificant, consisting mostly of minor corrections or updates, e.g., the UR “every example design data file” and the SysR “example data file”. Additionally, the mostly different pairs (i.e., the last two buckets in Fig. 5) were also trivially different, such as where the SysR’s field was empty (and so for analysis of future projects we would add an empty string to the stop-phrase list). To summarize the fit criterion field’s repetition over the projects, four-fifths of the UR–SysR pairs’ fields were repeated with no significant differences.

Finally, we examined correlation between the UR–SysR pair TF–IDF distance scores and the following variables:

1. UR and SysR description length;
2. UR and SysR document position (i.e., sequential IDs);
3. UR and SysR hierarchy depth, i.e., is SysR 1.1.1.1 more or less likely to be repeated from its UR than SysR 1.1?

### Table 11 Example UR–SysR TF–IDF distance scores and string “diff” analysis

<table>
<thead>
<tr>
<th>TF–IDF</th>
<th>Requirement Description Field (UR/SysR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.02</td>
<td>The <strong>user</strong> system shall be able to identify the manufacturer’s part number that applies to the subject item of production</td>
</tr>
<tr>
<td>0.12</td>
<td>The system shall be able to read the content of a file on a storage medium accessible via an address URL identified by the user</td>
</tr>
<tr>
<td>0.15</td>
<td>The <strong>user</strong> system shall be able to enable the <strong>user</strong> to specify a single file for conversion (single file mode)</td>
</tr>
<tr>
<td>0.19</td>
<td>The <strong>user</strong> system shall be able to determine indicate the success of the conversion process through a status information return</td>
</tr>
<tr>
<td>0.21</td>
<td>The <strong>user</strong> system shall be able to check all display a list of errors that arise from the processing of a new item create</td>
</tr>
<tr>
<td>0.25</td>
<td>The <strong>user</strong> system shall be able to convert one or more catalog files that are independent of the originating catalog management system from eOTD-r-xml to Modified Segment V (MSV)</td>
</tr>
<tr>
<td>0.4</td>
<td>The <strong>user</strong> system shall be able to respond to situations where existing display a list of items of supply that potentially require cancelation</td>
</tr>
<tr>
<td>0.84</td>
<td>The <strong>user</strong> system shall be able to identify the address of a file that contains the specification for the item support easy navigation through the hierarchy of accessible addresses</td>
</tr>
</tbody>
</table>

Key: strike-through: remove to form SysR; underline: insert to form SysR; plain style: commonality
4. UR and SysR hierarchical sibling position, i.e., is SysR 1.1.5 more or less likely to be repeated than 1.1.1?

5. Number of links to URs from the SysR.

No significant correlation for the first four variables was found in either project (the strongest Pearson’s coefficient was 0.31 for Project A’s SysR hierarchy depth and its description fields’ TF-IDF scores). However, Project B’s “number of links to URs from the SysR” was moderately correlated with the description fields’ TF-IDF distance scores (Pearson’s coefficient = 0.38). As could be logically derived, it appears that a significant proportion of Project B’s non-repeated UR–SysR pairs (i.e., the rightmost buckets in Figs. 3, 4, 5) occurred where the SysRs were linked to more than one UR. In other words, because there were more links between URs and SysRs than SysRs (98 vs. 60), more “related but not directly derived” UR–SysR pairs existed. In support of this conclusion is Project A’s weaker correlation in the same variables (Pearson’s co-efficient = 0.25) and its almost equal number of UR–SysR links and SysRs (55 vs. 52). Hence, there is less risk of wasting time due to the repetition defect when reading SysRs that are linked to more than one UR.

In summary, and to answer RQ2 in light of Table 10 (answering “how many projects?”) and Figs. 3, 4 and 5 (answering “how many requirements?”), we can say with confidence that redundancy-causing repetition between the textual fields of URs and SysRs has occurred in at least half of the projects in the context of this case study throughout the last 10 years. Based on the analysis of the sample requirement documents, and after confirmatory discussions with the survey participants, a conservative estimate is that, on average, one-third of the participants’ projects’ UR–SysR pairs exhibited redundancy-causing repetition. However, more than this proportion of a project’s SysRs will have been repeated from their URs, since a project’s SysRs are often linked to more than one UR, and so there are usually more UR–SysR pairs than SysRs. Furthermore, this estimate would be significantly higher if it only concerned the requirement’s fit criterion and rationale fields, which are often considered to be secondary to the description field. While we cannot generalize from these results to the wider context of IT software development, discussions with the participants (who have worked with many other non-defense organizations) indicate that similar results would be not be unlikely where UR and SysR documents are produced. (As Davies observed [7], it seems that many industry projects have one generic “requirements” documents containing an undifferentiated mixture of problem descriptions and solution prescriptions.)

### 4.2 RQ3: Why does UR–SysR repetition occur?

Logically, there can only be three possible modes of “blame” when a UR’s description is repeated across to a SysR:

1. The UR is defined too close to the solution space;
2. The SysR is defined too close to the problem space;
3. Despite analysis, the UR and the SysR are the same.

Before distributing the questionnaire, we interviewed three experienced consultants to ask why these modes of blame might occur. In total, five possible causes were elicited to explain why a UR or aSysR might be repeated to its associated SysR or UR:

- The stakeholder need was unknown;
- The solution to the need was unknown;
- There was not enough time to elicit the problem or derive a specification of the solution;
- Internal standards required both requirements sets (UR and SysR) regardless of the project’s context;
- Customer standards required both requirements sets (UR and SysR) regardless of the project’s context.

We then constructed a question asking the participants to rate the frequency of occurrence for each of these five causes. To mitigate researcher bias, the three consultants as well as the authors of this paper were asked to individually assess the adequacy of the causal categories, for which complete agreement was attained. An option to specify other causes was also added to the question. As with the previous questions, the ratings available to the participants were in the form of five ordinal categories (plus a “don’t know” option), ranging from the cause being applicable in none of their projects to all of their projects. Figure 6 summarizes the responses to this question by showing the distributions of the respondents’ ratings.

![Fig. 6 Causes of UR–SysR repetition: distributions of responses for the causes’ perceived frequency of occurrence](image-url)
The two “other” respondent-provided causes were:

- Ignorance of the conceptual difference between what a UR and a SysR should be;
- Laziness on the project stakeholders’ or requirements engineers’ behalf, i.e., they were not motivated to elicit or translate the appropriate requirements.

While these “other” responses provide invaluable insights into the causes of UR–SysR repetition, their frequencies cannot be compared to the others in Fig. 6, since other participants could have been reminded of their occurrence had they been shown them. (As such, for these two “other” causes, the “Don’t Know” totals represent no response, rather than explicit “Don’t Know” responses.)

Among the reported causes of UR–SysR repetition, “stakeholder need was unknown” (categorized under the first mode of blame) was significantly more commonly reported than “solution was unknown” (categorized under the second mode of blame). This is not surprising, since it is well known that stakeholders and requirements engineers tend to describe solutions rather than the problems to be solved by them [11]. (In such cases that lead to UR–SysR repetition, the requirements engineer has either duplicated or slightly rephrased the description of the proposed solution in the SysR into the supposed UR.) Stevens et al. [20] observe that practitioners often believe URs to be impossible to elicit, e.g., “Designers involved in making a sub-system, such as the engine for a car … often remark ‘users don’t understand what I’m building—there are no user requirements for my product’ … Under these circumstances, designers will often provide functionality which is perhaps not needed by the users”. Similarly, both Alexander’s paper entitled “Is There Such a Thing as a User Requirement?” [26], and van Lamsweerde’s textbook [25] provide arguments against the use of the term “User” to represent the problem description concept (as is a URs primary concern according to Table 1). Indeed, while end-users may not interact with components, those components should exist only to address stakeholder problems—even if the project was driven by means (i.e., innovation in IT) rather than by ends (i.e., problems), as Peppard et al. report is so often the case [82]. As such, requirements engineers are advised to investigate the underlying need for proposed solution capabilities by asking for their rationale [11, 20, 45], thereby assuring the pertinence of the proposed solution and enabling the evaluation of alternatives. If this good practice can be followed, then the leading cause of UR–SysR repetition (i.e., the only cause having the majority of respondents report its occurrence in at least half of their projects) could be avoided.

When a requirements engineer repeats a UR to a SysR (or vice versa), they do so out of choice or by constraint. Figure 6 shows that in this case study, the main (“by constraint”) reason for UR–SysR repetition was adherence to internal standards and processes that require UR and SysR documentation regardless of context such as project complexity. This cause was closely followed by a shortage of time (though this is likely a component cause for all RE defects), and then by the customer’s standards and processes requiring UR and SysR documentation regardless of context. A plausible explanation for the closeness of the “internal standard” and “customer standard” results is that the consulting company has adopted the same requirements standards and processes as their main customer, i.e., the UK MOD’s Acquisition Operating Framework (AOF), and so the small differences are likely to be caused by the existence of projects not owned by that customer.

Finally, there is the case where the UR seems to be equivalent to the SysR, i.e. the third mode of blame in Sect. 4.2. To illustrate, we refer to Clements and Bass’s anecdote [13] where a software project member prematurely specified the requirement of a particular database for data persistence. However, the requirement’s originator was a business manager in charge of the development staff who wanted to keep an under-worked database team busy. It could then be argued that the requirement encapsulates both a business problem and a software solution. However, the two should be separated into the need for the database team’s employment and the solution idea for data persistence, otherwise the requirement could be disregarded as rationale-less premature design, without describing that disregard’s consequences. Many of the guidelines in the sources cited by Table 1 make reference to such situations, e.g., Stevens [20] describes encountering a requirement for all engines on an airliner to be constantly powered, rather than for the airliner always having adequate power to fly. Nevertheless, it is plausible that a stakeholder could solely require a solution, i.e., if they (oddly) find intrinsic value in having all engines constantly powered, or in having a database, or in having trendy technology. Where the lack of a particular solution is stubbornly declared to be the problem, and only where adequate analysis of the underlying need has been attempted, UR–SysR repetition indicates less of a risk that the stakeholders would reject the system. However, it still causes requirements specification redundancy, limits innovation (by not exploring alternatives), and worse, risks that the system would not solve the underlying (and possibly changing) problems, despite initially satisfying the stakeholder(s) (ultimately leading to dissatisfaction). Hence, this third mode of blame should be widely discouraged.

4.3 RQ4: Is UR–SysR repetition a problem?

The questionnaire asked the participants if they consider UR–SysR repetition to be a problem, and Table 12 shows that most do.
The questionnaire then asked for the participants’ rationale for their responses in Table 12. Of those who believed it was a problem, 56% described generic redundancy-related consequences, e.g., the “time wasted creating and reading them,” and the “risk of discrepancies.” One such response described that the restatement of a UR into a SysR caused ambiguity and confusion. The next 31% of the responses described “deferral of systems thinking” and decision making to the developers, leading to suboptimal and delayed decisions in the development process, and in a high number of cases, scrap and rework. The remaining 13% of the responses can be summarized by the quote: “very often we define the what, but not the why”, which makes the quality of existing and new decisions difficult to ascertain. For example, one respondent described that selection among alternative design options was hindered since stakeholder preferences and rationale were not adequately described.

The participants who responded “no weakly” or “undecided” (in Table 12) were interviewed to explore their opinions. The most notable responses were:

- “Sometimes, URs and SysRs use the same terminology because it is the language that both the user and the developer use. In such cases, it is not necessarily helpful to seek to explain a clear UR in different terms for the SysR.”
- “If the project’s problem is not complex and well-tested solutions exist, then there is low risk in not translating the URs. However, the process we follow requires us to produce a UR and a SysR document regardless.”

The first quote presents an argument that UR-SysR repetition is not a problem if the project’s stakeholders “use the same terminology”, e.g., if they were all software developers, and so where the language dimension of difference between a UR and a SysR (Table 1) seems redundant. However, it does not follow that they should use the same terminology when writing URs and SysRs. While several of the guidelines in Table 1 recognize that the information [22] and “overall look, feel and structure” [35] within the different requirements documents will be similar, they stipulate that they should provide “different views for the same product” [22]. Similarly, Dorfman observes that system requirements may “closely resemble” subsystem requirements where they only need to be “allocated” to subsystems rather than translated (i.e., derived) [83]. However, this allowance concerns requirements within the solution space, rather than between the problem and solution space, which is where there is the most risk in “translating” the requirements. Hence, seeing the same terminology used between a UR-SysR pair indicates that the same information is being described, and hence that either the problem or the solution is not.

Follow-up discussions confirmed the URs were defined entirely in terms of user-system interactions, and so information about the problems to be solved by those interactions was missing. This risks the system being “unfit for purpose”, or at least being suboptimal compared to the possible alternatives. Overall, pursuing only descriptions from a technical (i.e., development or engineering) viewpoint neglects the essence of RE’s purpose.

The second quote presents the argument that low project risk and complexity permits not deriving a set of SysRs from the URs, instead repeating the SysRs from the URs in order to comply with a process that requires both URs and SysRs. Indeed, systems engineering standards and processes were derived by engineers experienced in complex projects such as satellites [8], which are arguably more risky and complex than the typical software development project. Instead, agile software development (which is rarely used for large or complex system development [21]) “shun[s] formal documentation of specifications” [79] and favors a “product backlog” consisting of “user stories”. These “high-level requirements” focus on the goals of the various user roles and their rationales, and defer description of the solution to “intensive communication” between the customers and the developers “before and/or during development” [79]. Overall, the risk involved in not creating SysRs (and hence where SysRs are entirely repeated from URs) seems negligible if:

1. The project is not risky from the developer’s viewpoint, i.e., there is little need for a contractually binding complete specification of system behavior;
2. “Intensive” communication between the developer and the customer is possible;
3. Creation of prototypes and/or frequent releases is economically viable;
4. The “optimal” solution to the problem is obvious and “tried and tested”.

Where these points apply, and where either external or internal standards require SysRs, then creating a set of SysRs closely resembling the URs for the sake of process compliance may be a “necessary evil” from the requirements engineer’s view. Indeed, since costs of deriving perhaps not-so-useful SysRs were avoided, the “Laziness” cause of UR-SysR repetition (Fig. 6) may actually benefit these projects. That being said, it is surely better to omit the SysR document than to repeat it from the URs, since repetition will lead to wasted time and inconsistency, despite

<table>
<thead>
<tr>
<th>Participant agreement on “Repetition between User Requirement and System Requirement pairs is a problem”</th>
</tr>
</thead>
<tbody>
<tr>
<td>No strongly</td>
</tr>
<tr>
<td>0 %</td>
</tr>
</tbody>
</table>
that it does not significantly risk the system being suboptimal. Unfortunately, the requirements engineer may not have the authority to make such a decision, and so UR–SysR repetition may prevail.

To conclude RQ4 (is UR–SysR repetition a problem?), the most concerning problem posed by repetition between a UR and a SysR is the increased risk that the stakeholders will not consider the system “fit for purpose” (as discussed in the summary of Table 1). The less consequential but more tangible problem is the increased redundancy within the requirements set. This will no doubt increase the reading and writing costs, in addition to increasing the risk of inconsistencies—a significant RE defect [25]. For an example of such an incurred cost, Gilb and Graham claim that requirements inspection occurs at approximately 10 words per minute [84]. According to that rate, inspecting the requirement descriptions of all UR–SysR pairs (55) in Project A (~790 words) would take 1 h and 19 min, of which, at the very least (TF–IDF distance score <0.1), ~24 min could be considered wasted due to repetition. (Reading costs are emphasized since the requirements will likely be read more than they will be written.) However, the cost of those 24 min is likely to be perceived as higher than normal, since they provide no reward (i.e., new information) and so demotivate the reader. Indeed, if even some of those 24 min occur sequentially, the reader may give up reading the UR and SysR requirements as a whole. Also, we are assuming that increases in reading effort are linear, but it could be argued that effort expended is higher for very similar UR–SysR pairs, since humans are inefficient at finding subtle differences that could nevertheless be important [3]. When considering the total cost of this redundancy, it should also be considered that a typical requirements inspection meeting involves several participants (including the customer on some occasions), and that each developer working on the project is expected to read the requirements to understand the problem they are to solve [13].

Caveat to the above conclusion is the hypothetical situation where either the UR or the SysR is conflated to perform the role of both describing the problem and also the abstract solution. There is also the perhaps more likely scenario where one of the documents (UR or SysR) contains a mixture of URs and SysRs [7]. Neither occurrence was apparent in this case study, but any conclusions reached after detecting such repetition would be affected. In the former scenario, the repetition would not indicate poor problem or solution exploration, and in the latter case, the “mode of blame” (in Sect. 4.2) is reversed.

4.4 RQ5: Can the problems be reduced (current/future projects)?

This section proposes guidelines for reducing the problems associated with UR–SysR repetition in current projects, i.e., where repetition already exists (1), and in future projects (2).

4.4.1 Reducing problems in current projects (where UR–SysR repetition exists)

Any increased risk that the product will not be accepted by the customer can only be reliably reduced through communication with the stakeholders (and possibly developers). Attempting to translate incorrect URs or SysRs without doing so (e.g., with the use of previous experience) may lead to the correct requirements, but, the risk will not have been reduced by doing so. Hence, this section will focus on the most tangible and therefore the most reducible problem caused by redundancy: increased reading time.

Software developers do not often read requirements documents to the extent that requirements engineers would like. For example, Spolsky writes that “the biggest complaint you’ll hear from teams that do write specs is that nobody reads them” [85]. Ambler adds to this that “some people naively think that developers do their work based on what is in the requirements document. If this was true, wouldn’t it be common to see printed requirements documents open on every programmers desk?” [86]. Indeed, with the common management mantra that “Weeks of coding can save hours of planning” [87], and the pressure placed on developers to “get on to the real work, designing and programming” [87], this is not surprising. The incentive to read requirements documents will be even less if there is a high degree of repetition between them. Reducing the non-value-added reading tasks can only contribute toward motivating developers to read requirements. Hence, we propose that UR–SysR links can be manually or automatically tagged with an “essentiality rating” to streamline the requirements reading process.

Essentiality ratings and tracks were first proposed to treat the problem of information overload in large documents and webpages [88]. The essence of the concept is that sections of content are marked with a rating of essentiality on a numerical 1–10 scale, and/or for a specific intended audience (i.e., a track), e.g., for a “UI developer”. Then, any nonessential or irrelevant content is filtered out for the reader. This concept could straightforwardly be applied to requirements documents to filter out repeated URs or SysRs from the set of requirements that the reader should read, where low UR–SysR pair TF–IDF distance scores could be translated to low essentiality scores. For example, the TF–IDF distance score of 0.3 would map to essentiality rating 3. (Note that the inverse is not applicable, since high essentiality cannot be determined by low similarity.) Alternatively, a requirements engineer who is authoring or reviewing a requirements document could manually estimate essentiality scores. For example, where
the UR is actually the same as the SysR (i.e., the third mode of blame described in Sect. 4.2), the requirements engineer would recognize that the UR is not essential reading if the SysR is read (or vice versa).

As aforementioned, manual inspection of this case study’s TF–IDF scores and the triviality of the actual differences indicated that the textual differences in UR–SysR pairs represented by TF–IDF distance scores lower than 0.3 were insignificant, (i.e., reading both requirements in such pairs does not add significant information). Hence, at the 0.3 TF–IDF distance threshold (above which non-repeated UR–SysR pairs existed), 43 % of Project A’s total UR–SysR description pairs and 16 % of project B’s UR–SysR description pairs could be considered “trivial” reading. (The percentages refer to the number of traceability links between URs and SysRs, as defined in Table 4.) If the requirements documents are being authored or reviewed, then these “trivial” UR–SysR pairs should be checked for missing information on the problem or the solution, among the other risks outlined in Sect. 4.3. Stated in terms of individual requirements rather than pairs of requirements, this “trivial reading” statistic is likely to be higher where there are more URs than UR–SysR traceability links. For example, 17 (56 %) of Project A’s 30 UR descriptions could be filtered out on the condition that the reader reads the set of SysRs. (Note that firstly, traceability between URs and SysRs is many-many [11], and so a UR is redundant only if it does not feature in a UR–SysR pair that contains non-trivial differences. Secondly, given the choice of reading a UR or a SysR in a repeated UR–SysR pair, the SysR should be chosen since it is likely to be the most recent version.)

The benefit of filtering UR–SysR pairs by essentiality ratings must be balanced with the risk of “false positives”, i.e., the risk that an important but marked-as-trivial difference is filtered out. Such a risk would not likely be entertained when reading requirements for contractual purposes, and hence the threshold could be set at 0.01 to filter out only the “identical” UR–SysR pairs. Conversely, false negatives do not pose such a significant problem, since the worst effect is that a repeated requirement is read, which is equal to the as-is situation, and has not cost the user any significant time due to the automated nature of the analysis. For example, the UR–SysR pair with the TF–IDF score of 0.4 in Table 11 was considered to be a repeated UR–SysR pair by human inspection, but is not classified as such by the 0.3 TF–IDF cutoff point. The consequence that the redundant requirement would not be filtered out for the reader was not considered to be a practical problem by the three interviewed stakeholders, despite that it is a valid direction of future research.

The appropriate TF–IDF distance threshold may slightly vary between projects, depending on factors such as the size of the vocabulary, the wordiness of the requirements, and for example, whether the project is safety critical. Hence, the requirements engineer should not assume that the threshold is a universal constant between projects, otherwise important information in URs or SysRs may be disregarded. Therefore, this process of determining the TF–IDF cutoff threshold (where thresholds >0.1 are desired) represents the majority of the time required to perform the automated UR–SysR repetition analysis, since the calculation of the TF–IDF scores takes seconds. Visualizing the differences between UR–SysR pairs, as shown in Table 11 (and as performed by our tool), is likely to be useful for determining the appropriate threshold. More rigorous approaches for determining optimal TF–IDF distance thresholds to improve conclusion validity are outlined by Falesi [61]. However, their relatively high cost of application (which threatens the usability and therefore the usefulness of the automated approach) means that in practical use, thresholds tend to be “chosen by common sense and without any rigorous criteria”—but not without reason [61].

4.4.2 Reducing problems in future projects (where UR– SysR repetition does not yet exist)

A great deal of UR–SysR repetition is caused by unknown stakeholder needs (the top cause of UR–SysR repetition in Fig. 6), and hence where both the URs and SysRs describe solutions. If stakeholders want software that solves problems in the most innovative and cost-effective way, and if requirements engineers want software that generates value rather than merely being of “good quality”, then resources from both sides should be committed to stakeholder requirements (i.e., needs) elicitation. To motivate stakeholders, evidence to show that poor requirements engineering and low stakeholder communication are top causes of project failure could be useful for motivating this change, e.g., [78, 89–91]. Motivating requirements engineers to elicit stakeholder needs rather than rushing to specify solutions, requires clarification that the “purpose of the requirements process is to add business value” [92], as well as fostering responsibility and reward mechanisms around this.

A common response from stakeholders when attempting to elicit their requirements is “I don’t know how to tell you, but I’ll know it when I see it” (IKIWISI) [93]. Consequently, Boehm proposes that requirements (in this context, URs) should focus on “how the new system will add value for each stakeholder” [93]. If stakeholders are unable to express their needs, then requirements engineers should attempt the use of well-established techniques for problem and goal elicitation, e.g., using scenarios, ethnography, interviews, questionnaires,
protocol/domain/task analysis, goal graph analysis, and so on [25]. If the IKIWISI problem is obscuring “what the solution should be”, then requirements engineers should attempt to elicit feedback from prototypes and early releases of the software [94]. Finally, market-driven software projects have slightly different challenges (e.g., less accessible stakeholders), for which the reader is referred to Karlsson et al.’s work [95].

Internal (i.e., the requirements engineer’s organization) standards or processes may stipulate that a UR and a SysR document be created regardless of the project’s context (i.e., its complexity, risk, “obviousness” of solution, etc.). Only well-argued cases for the project’s low complexity and a change in internal policy can reduce this source of UR–SysR repetition. If a customer’s standard requires both URs and SysRs regardless of the project’s context, then while the same argument applies, putting it into practice may be more difficult, due to the “customer is always right” mindset.

Finally, repetition between UR–SysR pairs caused by ignorance of the roles URs and SysRs play in the systems engineering life cycle should be treated by training the requirements engineers (or perhaps their trainers), especially with examples of good and bad practice rather than regurgitated “what not how” mantras. Crucially, motivation from management should be provided to maintain adherence to the true content of the standards (rather than misinterpreted versions sometimes found in textbooks or organizational training manuals). This will not happen quickly, since as aforementioned, requirements engineering (and indeed many other software “engineering” techniques [96]) are rarely considered a “top priority” in current software engineering practice.

5 Related work

As far as we are aware, Kovitz’s discussion of UR–SysR repetition (among a considerably larger set of software requirements engineering guidelines [14]) was the first treatment of the phenomena in the literature. Kovitz implies that many organizational standards mandate that their requirements engineers wastefully create UR and SysR documents comprised of near-identical repetition, as exemplified by requirement #7 in Table 2. Alexander later writes on the topic concluding that “you don’t need to be told (do you?) that a lot of time and paper is being wasted if your organisation is duplicating the requirements in that way” [15]. Alexander explains that in his experience, URs are often repeated from the SysRs because requirements engineers often believe that their only role is to describe system functionalities and qualities. As a consequence, he notes that the right-hand side of the V-model (verification and validation) is disrupted, since if both the SysRs and the URs describe a solution, then only verification can take place. Similarly, Davies notes that if only system requirements are captured, then “When it comes to the validation stage, the end-user may throw out the system as not fit for purpose, even if ‘It does exactly what it says on the tin’” [7].

While both Kovitz and Alexander imply that UR–SysR repetition has frequently occurred (e.g., “too often led tired engineers to write…” [15]), they provide neither an investigation nor an analysis of the real-world problem and its causes. Furthermore, neither Kovitz nor Alexander mention repetition between the different composite fields of a requirement other than the description field (i.e., the “main” field). This is significant since complete duplication in one field does not infer that the UR or SysR is completely redundant. Finally, both Kovitz and Alexander propose ways of avoiding UR–SysR repetition. Kovitz [14] proposes that requirements should be organized by their subject matter rather than by their “level of detail”. However, this neglects the issues that creating separate UR and SysR documents address, i.e., for the different reasons outlined in Table 1. On the other hand, Alexander proposes that URs should be renamed to “business problems”, and SysRs to “system solutions”. This seems more semantically correct in light of the definitions in Table 1 as well as the criticisms in the literature of the term “User” in “User Requirement” [25, 26]. However, the “business” term is also a restrictive subtype of stakeholder requirement, and “solution” implies that the problem will be (fully) solved, rather than treated [97]. Finally, Kovitz and Alexander’s discussions are over a decade old, and so prior to this research there was a need to understand the pertinence of the UR–SysR repetition problem in the current industry environment.

Juergens et al. [3] analyze the degree of “copy&paste” clones within requirements documents with their ConQAT tool. They report an average “blow-up” (i.e., “the ratio of the total number of words to the number of redundancy free words”) of 13.5 % across 28 software requirements documents from different industries. In their conclusion, they propose that requirements document blow-up of more than 5 % should be considered “as a warning signal for potential future problems”. However, ConQAT does not parse requirement documents into individual requirements (nor their traceability links), but rather searches through a plaintext requirements document for contiguous fixed-length sequences of words (in their study, 20 words) that occur more than once. Thus, their work is concerned with verbatim duplication of requirement document text, rather than requirements “which have been copied but slightly reworded in later editing steps” [3], which we have found to be very common in UR–SysR repetition. Furthermore,
their work does not consider traceability to other requirements, nor repetition between a project’s different requirements documents, and is therefore unable to model nor analyze UR–SysR pairs.

Due to the commonality of natural language specifications and documentation in software projects, natural language processing techniques have been applied to problems in various areas of software engineering. For example, Runeson et al. [57] successfully detect two-thirds of duplicate software defect reports using vector space representation and term frequency weighting (i.e., TF rather than TF–IDF). Closer to our work, numerous researchers have tackled various requirements engineering problems with similar techniques, as the remainder of this section discusses.

Natt och Dag et al.’s ReqSimilie tool [98] suggests traceability links between requirements based on the assumption that requirements containing similar words are likely to be related since requirements engineers strive for consistent use of terminology. Regardless of the validity of that assumption for URs and SysRs, the focus of their work is the opposite to ours. ReqSimilie does not consider existing traceability links, and attempts to find requirements worth linking to, whereas our approach assumes traceability links exist and attempts to find those that do not add different information. Hence, using ReqSimilie, it is not possible to know the degree of repetition between UR–SysR pairs, since the information that defines pairs of URs and SysRs is disregarded. Furthermore, ReqSimilie’s assumption that requirements are stored in a database rather than in documents means that it would not be easily usable by the majority of requirements engineers who use word processing or spreadsheet software, as is the case in our case study, and in numerous other surveys on RE practice [52, 99].

Similar to the ReqSimilie tool is Cleland-Huang et al.’s “Poirot Trace Maker” [100], which automatically suggests traceability links between requirements, design, and code artifacts, (e.g., requirement text to UML sequence diagram messages to Java method names). Poirot Trace Maker also uses vector space representation to compute the degree of artifact similarity, and so it is similar to ReqSimilie in technique but not in intent. Interestingly, Cleland-Huang et al. cite Hull et al.’s example UR–SysR pair (#1 in Table 2) as an example of requirements that cannot be automatically traced, since only the fairly generic terms (the, shall, to, vehicle) are shared between them. It is therefore implied that repetition of rare terms between URs and SysRs is advantageous for the sake of automated traceability. However, this would violate the guidelines on good URs and SysRs provided in Table 1, e.g., since the language used to describe a stakeholder’s problem and an engineer’s solution should be different. Instead, Cleland-Huang et al. propose that the use of Hull et al.’s “satisfaction arguments” [11] (i.e., a description of why the SysR satisfies the UR) could increase the recall rate, since satisfaction arguments tend to contain terminology from both the URs and SysRs. We are not optimistic about this from this paper’s viewpoint (repetition), or from a pragmatic viewpoint, since:

1. A significant amount of Hull et al.’s satisfaction arguments are repeated from the UR and the SysR, introducing redundancy and hindering modifiability;
2. Satisfaction arguments are a form of manual traceability between a URs and SysRs, and so by the time a SysR could be automatically linked by using the text from a satisfaction argument, it is already manually linked.

If repetition between URs and SysRs is to be avoided, then automated traceability between them requires “problem → solution” knowledge. This could be inferred from traceability links within previous projects, or could be explicitly and ontologically defined (e.g., encoded with “means-end” links in GRL [41]). For example in the context of Hull et al.’s UR–SysR pair (#1 in Table 2), “power to all wheels” is-a-solution-to “deployment on wet mud”, and “wet mud” is “terrain type 4A” (the latter should ideally be defined within the UR for the sake of completeness).

Lami’s QuARS tool [101] automatically detects linguistic defects that could cause ambiguity, understandability, or completeness problems. However, QuARS is concerned with the quality of single requirement statements, and so completeness is assessed only in the context of each individual requirement, as opposed to the completeness of a SysR set relative to its coverage of a UR set. Similarly, Park et al. [102] propose an automated requirements analysis system to detect ambiguity and incompleteness in requirements documents. Closer to our problem is Park et al.’s claim that their system can trace dependency and reduce inconsistency between “a sentence in a high-level document and a sentence in a low level document”. However, in the examples Park et al. use to explain their claim, the “level” of a requirement refers to its level of decomposition rather than its domain (problem or solution), or other dimension of UR–SysR difference listed in Table 1. As such, Park et al.’s claims are not applicable to SysR documents derived from UR documents, since “unlike decomposed requirements, the statements of the derived requirements are different from those of the original requirements”, to quote Sage and Rouse [28].

Finally and most recently, Ferrari et al. [103] explore the use of the Sliding Head-Tail Component clustering algorithm (rather than the vector space model of similarity) to propose a new requirement document structure, in order to optimize the structure’s “requirements relatedness” and
“sections independence” qualities. Despite not being directly applicable to this paper’s topic, it would be interesting to apply their technique to combine UR and SysR documents, e.g., to automatically generate a requirements document that presents both sets of requirements, structured by their relatedness.

6 Conclusion and future work

Duplication within requirements documents is clearly considered to be a quality defect; international standards require that within “the set of stakeholder, system, and system element requirements … requirements are not duplicated” [22], and that software requirements “not be redundant” [4]. In this paper, we have proposed that repetition between URs and SysRs can also indicate a more serious defect: incompleteness of the requirement set. In other words, we find UR–SysR repetition concerning since it indicates that one set of requirements (URs or SysRs) is not complete. This is especially troubling where the solution description is repeated, since Jackson [36] makes it quite evident that the main concern in RE should be describing the application domain (i.e., problem) rather than the machine (i.e., solution). In other words, good engineers can derive solutions from problems, but the converse is less likely.

Aside from anecdotes and examples, there has been no published investigation to show if and why repetition between URs and SysRs occurs. We have presented a case study with the intention of adding to the “chain of evidence” [64]. We found that 75 % of the survey participants had seen UR–SysR repetition in at least half of their projects (Sect. 4.1.1). Then, we found that on average over our sample projects, one-third of the UR–SysR pairs had description fields that contained significant repetition (Sect. 4.1.2), while the UR–SysR pairs’ fit criterion and rationale fields exhibited roughly twice as much repetition. Finally, our survey found that unknown stakeholder needs combined with the internal (i.e., the requirements engineer’s) organization’s context-independent stipulation for UR and SysR documentation were the most popular reasons for UR–SysR repetition (Sect. 4.2).

Based on the results of our research questions, we propose the following recommendations:

- Despite that there will likely be a significant amount of repetitious UR–SysR pairs within UR and SysR documents (especially within the non-primary attributes such as the rationales or fit criteria), stakeholders including developers should still read both, since numerous non-trivially different and informative UR–SysR pairs are also likely to exist. UR–SysR pairs where the UR is related to more than one SysR (or vice versa) are more likely to provide information to make their reading as a pair worthwhile. Where a UR has only one related SysR, the most recent of the two requirements should be read.

- Requirements engineers should ensure that stakeholder needs have been adequately elicited and analyzed if repetition between UR–SysR pairs exists. Not understanding the problem is a significant and well-established software project failure risk, that is also known to cause problems such as scrap and rework, poor innovation, suboptimal decisions, or value failure.

- Training on the role of URs and SysRs should be provided where repetition exists due to ignorance of their roles in the engineering process. Perhaps more important is that management should motivate and monitor adherence to RE standards (but not for rigor on principle).

- Checking for UR–SysR repetition can be performed with little effort using our software tool [16], and hence could be built into future requirements quality inspections. Where a significant degree of UR–SysR repetition exists, the tool could be used to streamline the reading process by filtering out trivial URs or SysRs.

Finally, it is important to remark upon the applicability of this paper to software engineering practice outside of the defense industry. Firstly, as Table 1 exemplifies by having only one source from the defense industry, the recommendation to distinguish between URs and SysRs is not limited to defense, but is advocated in general software and systems engineering practice. Indeed, in Davies’ article “Ten Questions to Ask Before Opening the Requirements Document” (targeted to the general field of systems engineering), the first question is “Is it a User Requirement, [or] a System Requirement…?” [7], while in Wiegers’ article “10 Requirements Traps to Avoid” (targeted to the general field of software requirements engineering, i.e., not defense or systems), the first trap is “that project stakeholders refer to ‘the requirements’ with no qualifying adjectives … [and] as a consequence, important stakeholder expectations might go unstated and unfulfilled” [104]. So, if one can agree that the concepts behind URs and SysRs are applicable to software and system engineering, then the main question of this paper’s applicability becomes “how frequently are software systems engineered?” On this topic, and in the face of the growing Agile software development community, software engineering expert Demarco [96] recently wrote that “I’m gradually coming to the conclusion that software engineering is an idea whose time has come and gone. I still believe it makes excellent sense to engineer software. But that isn’t exactly what software engineering has come to
mean”. Indeed, as an indication of popularity, according to “Google Trends” [105] the term “user requirement” (or “stakeholder requirement”) receives six times less search interest than Agile’s equivalent “user story” term in 2015, whereas they were roughly equal in 2006. However, search interest in the former term has remained approximately constant over the last decade, and the use of systems engineering standards such as ISO 29148 (one of the most authoritative proponents of URs and SysRs) is still adopted in many industries where there is high complexity, expenditure, and risk (e.g., in aero, automotive, or military engineering) [21, 25]. (Common reasons for not adopting Agile include costs of frequent communication between end-users and developers [79], or a need for verification prior to coding, among others discussed in [25, 106].)

In a comparison of Systems Engineering with Agile, Turner concludes that “there are still no silver bullets [107]” [108]; the key concerns behind traditional “UR–SysR” requirements engineering (summarized in Table 1) still exist in modern software development. Merisalo-Rantanen et al. [109] even argue that much of modern techniques are largely a case of “Old Wine in New Bottles”. Indeed, an Agile user story using Cohn’s [94] template of “As a <user type>, I want to <goal> so that <reason>” could be interpreted such that the <reason> field maps to the concerns of URs, while the <goal> field maps to user-task-oriented SysRs. Interestingly, some repetition between these “fields” of user stories is apparent, even in examples on Cohn’s website [110].

Ultimately (and as the division of Sect. 4.4 into current practice and future practice makes clear), the ideal goal is to not need a tool for assessing the quality of requirements, since the requirements processes should be optimized for each software development context. However, in the past and most likely in the future, people will make mistakes and apply techniques in contexts ill-suited to them [111], either by choice or due to constraints such as organizational standards. Furthermore, the current and future trend where “commercial off-the-shelf (COTS) capabilities [i.e., solutions] determine requirements” [108], serves to increase the importance of ensuring that those capabilities are related to real stakeholder needs not described in terms of those capabilities. Overall, this paper’s topic is pertinent in the current and foreseeable future state of practice, despite that it would not be in an ideal world.

As future work, a number of interesting research questions were identified during the project:

- Is there a correlation between software project success (or the amount of rework or ad hoc communication required to make it successful) and the compliance of URs and SysRs to requirements engineering standards?
- How common is UR–SysR repetition (or even the production of UR and SysR documents) in software projects within different industry contexts (i.e., non-defense)?
- Is the degree of UR–SysR repetition correlated with the number of authors creating the project’s requirements documents? In other words, is better UR and SysR separation achieved when different authors write them?
- Can a greater number of trivial differences be identified in UR–SysR pairs by extracting and comparing semantic information, such as by using domain ontologies and lexical databases (e.g., Princeton’s WordNet), or by comparing “Parts of Speech” (as are extracted from requirements in [112])? This would explore whether RE authors purposely replace SysR terms with synonymous terms to form a UR that appears to be different.
- Can machine learning techniques (e.g., a naïve Bayesian classifier) be used to better classify UR–SysR pairs as to whether useful information is added by a UR–SysR link?
- What are the implications of repetition of natural language within requirements engineering models, e.g., between TROPOS goals and plans, as is visible in [113]?
- How effective (i.e., credible [61]) is the proposed approach at filtering out trivial URs or SysRs (the answer to RQ5) in a wider variety of requirements documents? That is, how many trivial UR–SysR pairs (assessed manually by stakeholders) are not filtered out, and how many non-trivial UR–SysR pairs are filtered out?

Acknowledgments The authors are grateful to LSC Group for allowing data collection and publication of results. Special thanks are owed to LSC Group’s Chris Lambert, Dr. James Nyambayo, and Dr. Ann Meads for the insightful discussions on the topic. Warm thanks are due to the reviewers of this paper whose suggestions resulted in the clarification of many points, and to both Springer’s editors and Sachi Jain for proofreading and copy-editing.

References

Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14:131–164. doi:10.1007/s10664-008-9102-8
1. Analysing Repetition Between User Requirements and System Requirements Documents

1.1 Step 1: New GoalViz Project

A. Open the GoalViz.jar file from the extracted \textit{zip archive}.

B. File $>$ New $>$ Project $>$ Choose somewhere to store the GoalViz project database file & press OK.

C. Add two folders (using the ‘+Folder’ icon at the bottom left of the screen). They can be called anything, but I have chosen ‘URs’ and ‘SysRs’.

   a. You can add two new requirements (the ‘+Speech Bubble’ icon), and then drag one requirement to the other in the tree to see the traceability visualisation capabilities of GoalViz.
1.2 Step 2: Import Requirements Statements

D. Import/Export > Import Requirements CSV (Excel)

E. Choose the CSV file containing the requirements statements (MS Excel can save as CSV files) – there is no requirement for structuring the requirements in a pre-defined order.
F. Select the folder in GoalViz to insert the requirement statements from the CSV file.

G. Select the row that came from the CSV file that contains the heading fields (i.e., the row that says something like ‘ID, Description, Rationale,’).

H. Choose the GoalViz requirement field that matches each of the columns from the CSV file.
E.g., the ‘OBJECT IDENTIFIER’ field came from CSV file’s field heading row, and it specifies a global (i.e., references in other documentation) unique identifier for the requirement statement. The ‘SOURCE USER REQUIREMENT’ field from the CSV file contained the references (i.e., the global ID’s) to the requirement(s) in the other requirements document.

1. Confirm that the structure and fields have been imported properly.
J. Import the other requirement document(s), i.e., if that was the User Requirements Document, now import the System Requirements Document.

K. Import/Export > Reconstruct Derived Traces — this parses the textual links to and from the imported requirements (as defined in the GoalViz field requirement_imported_id_global_relationships).

1.3 Step 3: Run UR-SysR Repetition Analysis

L. Analysis > Analyse Traced Requirement Repetition

M. Specify the fields to compare between the traced requirements:
The results of the analysis will be shown in the ‘Report’ tab after a few seconds (the tool has been tested and optimised for requirement sets >1000 statements). Requirement statements can be filtered and sorted.
Explore the other options/analysis methods in the tool!
Appendix H - Survey to company 1

Survey of RR Employees on Software Requirements & Benefit Realisation

Page 1 of 2

**Aim:** To capture preliminary evidence on the strategic alignment of software requirements, as part of a PhD research project.

**Questions:** 13

**Estimated Completion Time:** ~10 minutes.

---

1. Which RR function do you work for?

   - Select an answer

   - If you selected Other, please specify:

---

### Section 1/3: Your Experience in Software Projects

Please limit your responses to the last 10 years of your career, i.e., from 2003 onwards.

2. How many years of experience do you have in the development/engineering/production of software?

   - 0 years
   - ≤5 years
   - ≤10 years
   - ≤15 years
   - ≤20 years
   - ≤25 years
   - ≤30 years
   - >30 years

3. How many software development projects have you contributed toward in any way (last 10 years)?

   - 0 projects
   - ≤5 projects
   - ≤10 projects
   - ≤15 projects
   - ≤20 projects
   - ≤25 projects
   - ≤30 projects
   - >30 projects

4. How many of these projects do you know of that are now in use (or were) by the end-users?

   - Almost none
   - Some
   - Around half
   - Most
   - Almost all
   - [don't know OR n/a]

   a. For how many of these projects (from q4) do you have any knowledge of the usefulness/pertinence of the implemented requirements?

      - Almost none
      - Some
      - Around half
      - Most
      - Almost all
      - [don't know OR n/a]

   b. For how many of these projects (from q4.a) do you have any knowledge of the costliness of the implementation of the requirements?

      - Almost none
      - Some
      - Around half
      - Most
      - Almost all
      - [don't know OR n/a]

   c. How many of these projects (from q4.a) do you know of that achieved their intended benefits?

      - Almost none
      - Some
      - Around half
      - Most
      - Almost all
      - [don't know OR n/a]
Section 2/3: The Pertinence of Functional & Non-Functional Requirements

Reminder: Please limit your responses to the last 10 years of your career.

5. On average over the projects you know about, how many of the implemented Functional Requirements (FRs) were redundant (i.e., their implementation was unused or scrapped)?

- Almost none of the FRs
- Some of the FRs
- Around half of the FRs
- Most of the FRs
- Almost all of the FRs
- [don't know OR n/a]

Reminder: FRs specify what actions the software should be able to perform (e.g., should be able to print reports).

6. On average over the projects you know about, how many of the implemented Non-Functional Requirements (NFRs) were either wasteful or inadequate?

- Almost none of the NFRs
- Some of the NFRs
- Around half of the NFRs
- Most of the NFRs
- Almost all of the NFRs
- [don't know OR n/a]

Reminder: NFRs specify qualities of the software such as reliability or performance.

1. Example of a wasteful non-functional requirement: "The heart rate monitor should be accurate to within 0.01bpm". This level of accuracy provides no extra value in the intended context of use (hospital), but does increase costs.
2. Example of an inadequate non-functional requirement: "The heart rate monitor should be accurate to within 30bpm". This level of accuracy is inadequate for the intended context of use (hospital), and increases costs in re-work or re-purchasing.

7. In your opinion, how many of these wasteful implementations (from q5 & q6) could have been avoided (e.g., through better analysis or communication)?

- Almost none of the wasteful implementations
- Some of the wasteful implementations
- Around half of the wasteful implementations
- Most of the wasteful implementations
- Almost all of the wasteful implementations
- [don't know OR n/a]

8. Briefly provide your opinion on why wastage occurred (i.e., why the software was not as 'useful' as hoped), if any did:

(Optional)

---

9. On average over the projects you know about, how costly were these wasteful implementations (from q5 & q6)?

The redundant features or wasteful/inadequate quality accounted for ...

- Almost none of the project's costs
- Some of the project's costs
- Around half of the project's costs
10. If these wasteful implementations (from q5 & q6) were costly, which were more costly?

- Non-functional requirements (wasteful/inadequate)
- Functional requirements (unused)
- About equally split (between NFRs & FRs)
- [don't know OR n/a]

  If you chose non-functional requirements, were they mostly:
  - Wasteful
  - About equally split (wasteful/inadequate)
  - Inadequate

11. How frequently was it the case that:

<table>
<thead>
<tr>
<th></th>
<th>almost none of the projects</th>
<th>some of the projects</th>
<th>around half of the projects</th>
<th>most of the projects</th>
<th>almost all of the projects</th>
<th>[don't know OR n/a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Requirements were quantitatively defined (e.g., with fit criterion or test cases)?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>b. Requirements tended to prescribe technical solutions rather than describe problems?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Section 3/3: Business Benefits & Objectives

Reminder: Please limit your responses to the last 10 years of your career.

12. How frequently was it the case that:

<table>
<thead>
<tr>
<th></th>
<th>almost none of the projects</th>
<th>some of the projects</th>
<th>around half of the projects</th>
<th>most of the projects</th>
<th>almost all of the projects</th>
<th>[don't know OR n/a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. You felt some responsibility for ensuring that the software project provided value-for-money?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>b. You had an understanding of how much business value the software requirements would create (‘line of sight’)?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>c. End-users/stakeholders were easily contactable to clarify/explain their requirements?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>d. Software project releases were late due to delivering functionality not considered to be essential to the end-user or business?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
13. If links (relationships/traceability) between software requirements and their benefits/objectives were made, was it (or if it wasn’t, do you think it would have been):

<table>
<thead>
<tr>
<th></th>
<th>almost none of the projects</th>
<th>some of the projects</th>
<th>around half of the projects</th>
<th>most of the projects</th>
<th>almost all of the projects</th>
<th>[don’t know OR n/a]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a.</strong> Useful (e.g., in prioritising requirements, understanding rationale)?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td><strong>b.</strong> Easy to create (i.e., to establish explicit traceability)?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td><strong>c.</strong> Easy to understand (i.e., to comprehend how the requirement would achieve the benefit)?</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

There are no more mandatory questions, please click continue to submit your responses.
Survey of RR Employees on Software Requirements & Benefit Realisation

Page 2 of 2 - A few questions about you

Thanks for submitting your responses, the mandatory part of the survey is now complete, and you may now close the browser if you wish.

(If you have the time, please complete the following questions about your experience so that we can better contextualise the results. Estimated Time to Completion: 5 minutes)

Section 5/6: Your Professional Background

14 Describe your involvement in the following software engineering activities in the last 10 years:

<table>
<thead>
<tr>
<th>Experience</th>
<th>None</th>
<th>Some</th>
<th>Substantial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Case (such as cost-benefit analysis)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>User Requirements (description of user needs)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>System Specification (abstract description of solution)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>System Design (high-level architecture)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Component Design (low-level class/method design)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Component Construction (programming)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Component Test (checking against the component design)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>System Integration Test (checking against the system design)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>System Test (checking against the system specification)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Acceptance Test (checking against the user requirements)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Software Engineering Management</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Software Engineering Process Development</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

15 Describe your involvement in the following requirements engineering activities in the last 10 years:

<table>
<thead>
<tr>
<th>Experience</th>
<th>None</th>
<th>Some</th>
<th>Substantial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Elicitation (discovering needs &amp; constraints)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Requirements Analysis (refining needs &amp; constraints)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Requirements Specification (documenting the needs precisely)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Requirements Verification (ensuring completeness, clarity, etc.)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Requirements Management (coordinating &amp; documenting the above)</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Section 6/6: Closing Questions
Would your answers have been significantly different if the survey had asked for your experience from:

<table>
<thead>
<tr>
<th>Answers concerning wasteful/redundant requirements would be...</th>
<th>better</th>
<th>no different</th>
<th>worse</th>
<th>[don't know OR n/a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your last 5 years rather than the last 10 years?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Your entire career rather than the last 10 years?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any other comments/thoughts relating to your responses or the survey:

Please leave your email address if you would not mind having a follow up discussion to further your answers:

Your anonymity in the output of this survey will be unaffected by leaving your email address.

Any queries can be sent to: R.D.J.Ellis-Braithwaite@iboro.ac.uk

Please press continue to finish the survey, thanks for your time!
Appendix I - Survey to company 2

Survey of LSC Employees on Software Requirements & Benefit Realisation

Page 1 of 2

Aim: To capture preliminary evidence to guide research on modelling the strategic alignment of software requirements (for a PhD research project).

Total Questions: 21
Estimated Total Completion Time: No longer than 20-25 minutes.

Section 1/6: Software Projects

Please limit your responses to the last 10 years of your career, i.e., from 2003 onwards.

1. How many software development projects have you contributed toward in any way (last 10 years)?
   - 0 projects
   - ≤5 projects
   - ≤10 projects
   - ≤15 projects
   - ≤20 projects
   - ≤25 projects
   - ≤30 projects
   - >30 projects

   *Software development projects* includes software sub-projects of non-software projects, such as an engine control system (software) for a new engine (not software).

2. Have these projects been for:
   - Mostly military customers
   - Mostly civil customers
   - About equally split (military/civil)

3. How many of these projects do you know of that are now in use by the end-users (or were)?
   - Almost none
   - Some
   - Around half
   - Most
   - Almost all
   - [don't know OR n/a]

   a. For how many of these projects (from q3) do you have any knowledge of the usefulness/pertinence of the implemented requirements?
      - Almost none
      - Some
      - Around half
      - Most
      - Almost all
      - [don't know OR n/a]

   i. For how many of these projects (from q3.a) do you have any knowledge of the costliness of the implementation of the requirements?
      - Almost none
      - Some
      - Around half
      - Most
      - Almost all
      - [don't know OR n/a]

   b. How many of these projects (from q3) do you know of that achieved their intended benefits?
      - Almost none
      - Some
      - Around half
      - Most
      - Almost all
      - [don't know OR n/a]

Section 2/6: The Pertinence of Functional & Non-Functional Requirements

Reminder: Please limit your responses to the last 10 years of your career.

4. On average over the projects you know about, how many of the implemented Functional Requirements (FRs) were redundant (i.e., their implementation was unused or scrapped)?
   - Almost none of the FRs
   - Some of the FRs
   - Around half of the FRs
   - Most of the FRs
   - Almost all of the FRs
   - [don't know OR n/a]

Reminder: FRs specify what actions the software should be able to perform (e.g., should be able to print reports).
5. On average over the projects you know about, how many of the implemented non-functional requirements (NFRs) were either wasteful or inadequate?

- Almost none of the NFRs
- Some of the NFRs
- Around half of the NFRs
- Most of the NFRs
- Almost all of the NFRs
- [don’t know OR n/a]

Reminder: NFRs specify qualities of the software such as reliability or performance.

1. Example of a wasteful non-functional requirement: "The heart rate monitor should be accurate to within 0.01bpm". This level of accuracy provides no extra value in the intended context of use (hospital), but does increase costs.
2. Example of an inadequate non-functional requirement: "The heart rate monitor should be accurate to within 30bpm". This level of accuracy is inadequate for the intended context of use (hospital), and increases costs in re-work or re-purchasing.

6. On average over the projects you know about, how costly were these requirements errors (from q4 & q5)?

The redundant features or wasteful/inadequate quality accounted for ...

- Almost none of the project’s costs
- Some of the project’s costs
- Around half of the project’s costs
- Most of the project’s costs
- Almost all of the project’s costs
- [don’t know OR n/a]

7. If these requirements errors (from q4 & q5) were costly, which were more costly?

- Non-functional requirements (wasteful/inadequate)
- Functional requirements (unused)
- About equally split (between NFRs & FRs)
- [don’t know OR n/a]

If you chose non-functional requirements, were they mostly:

- Wasteful
- About equally split (wasteful/inadequate)
- Inadequate

8. In your opinion, how many of these wasteful implementations (requirements errors from q4 & q5) could have been avoided (e.g., through better analysis or communication)?

- Almost none of the wasteful implementations
- Some of the wasteful implementations
- Around half of the wasteful implementations
- Most of the wasteful implementations
- Almost all of the wasteful implementations
- [don’t know OR n/a]

Briefly provide your opinion on why wastage occurred, if any did: (Optional)

9. How frequently were requirements left unimplemented that would have been more valuable (better cost/benefit) than those which were implemented?

- In almost none of the projects
- In some of the projects
- In around half of the projects
- In most of the projects
- In almost all of the projects
- [don’t know OR n/a]

Section 3/6: Business Benefits & Objectives

Reminder: Please limit your responses to the last 10 years of your career.
10. How frequently was it the case that:

<table>
<thead>
<tr>
<th>a. The system's end-users were easily contactable to clarify/explain their requirements?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b. You felt some responsibility over ensuring that the software project provided value-for-money?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c. The system-owner's business objectives for the projects were available?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d. The system-owner's higher-level business objectives (i.e., not specific to the projects) were available?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e. The system-owner's business objectives were quantitatively defined?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>f. The software requirements were quantitatively defined (e.g., with fit criterion)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>g. The software requirements were explicitly traced (i.e., relationships made) to their proposed benefits (e.g., to business objectives)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

11. If traceability from requirements to benefits/objectives was made available, was it (or if it wasn't, would it have been):

<table>
<thead>
<tr>
<th>a. Useful (e.g., in prioritising requirements)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b. Easy to understand (i.e., to comprehend how the requirement would achieve the benefit)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c. Easy to create (i.e., to establish explicit traceability)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>almost none of the projects</td>
</tr>
</tbody>
</table>

**Section 4.6: Requirements Documents**

Reminder: Please limit your responses to the last 10 years of your career.

12. Have you seen repetition between a project's User Requirement descriptions and System Requirement descriptions?

- Almost none of the projects
- Some of the projects
- Around half of the projects
- Most of the projects
- Almost all of the projects
- [don't know OR n/a]

In this context, an example of repetition is "system shall be able to x" and "user shall be able to x", since both requirements prescribe that x should be possible.

13. If you have seen such repetition, do you think it is an issue?

- Yes, strongly
- Yes, weakly
- Undecided
- No, weakly
- No, strongly

Briefly describe why you think it is, or is not an issue? (Optional)
14. If you have seen repetition, why do you believe it occurred?

<table>
<thead>
<tr>
<th></th>
<th>almost none of the projects</th>
<th>some of the projects</th>
<th>around half of the projects</th>
<th>most of the projects</th>
<th>almost all of the projects</th>
<th>[don't know OR n/a]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15. If other (question 14), please describe: (Optional)

The next (and last) page is much shorter than this one - only 6 quick questions (about you) left!
Survey of LSC Employees on Software Requirements & Benefit Realisation (p9)

Page 2 of 2 - A few questions about you

Estimated Time to Completion: 5 minutes

Section 5/6: Your Professional Background

16. How many years of experience do you have in the development/engineering/production of software?
   - 0 years
   - ≤5 years
   - ≤10 years
   - ≤15 years
   - ≤20 years
   - ≤25 years
   - ≤30 years
   - >30 years

17. Describe your involvement in the following software engineering activities in the last 10 years:

<table>
<thead>
<tr>
<th>Experience</th>
<th>None</th>
<th>Some</th>
<th>Substantial</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Business Case (such as cost-benefit analysis)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b. User Requirements (description of user needs)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c. System Specification (abstract description of solution)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d. System Design (high-level architecture)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e. Component Design (low-level class/method design)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>f. Component Construction (programming)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g. Component Test (checking against the component design)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>h. System Integration Test (checking against the system design)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>i. System Test (checking against the system specification)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>j. Acceptance Test (checking against the user requirements)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>k. Software Engineering Management</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>l. Software Engineering Process Development</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

18. Describe your involvement in the following requirements engineering activities in the last 10 years:

<table>
<thead>
<tr>
<th>Experience</th>
<th>None</th>
<th>Some</th>
<th>Substantial</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Requirements Elicitation (discovering needs &amp; constraints)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b. Requirements Analysis (refining needs &amp; constraints)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c. Requirements Specification (documenting the needs precisely)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d. Requirements Verification (ensuring completeness, clarity, etc.)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e. Requirements Management (coordinating &amp; documenting the above)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Section 6/6: Closing Questions

19. Would your answers have been significantly different if the survey had asked for your experience from:

<table>
<thead>
<tr>
<th></th>
<th>Answers concerning wasteful/redundant requirements would be...</th>
<th>Answers concerning repetition in requirement descriptions would be...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>better</td>
<td>no different</td>
</tr>
<tr>
<td>a. Your entire career rather than the last 10 years?</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>b. Only military projects?</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

20. Any other comments/thoughts relating to your responses or the survey: (Optional)


21. Please leave your email address if you would not mind having a follow up discussion to further your answers: (Optional)

---

Your anonymity in the output of this survey will be unaffected by leaving your email address.

Any queries can be sent to: R.D.J.Ellis-Braithwaite@lboro.ac.uk
Please press continue to finish the survey, thanks for your time!