The effectiveness of argumentative approaches to the design of software

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

Additional Information:


Metadata Record: https://dspace.lboro.ac.uk/2134/33159

Publisher: © Georgios P. Iliadis

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/

Please cite the published version.
Please note that fines are charged on ALL overdue items.

LOAN COPY
The effectiveness of argumentative approaches to the design of software

by

Georgios P. Iliadis

A Doctoral Thesis Submitted in Partial Fulfilment of the Requirements for the Award of Doctor of Philosophy of Loughborough University

May 1999

© Georgios P. Iliadis, 1999
Abstract

This thesis investigates the potential of design rationale to support software designers working on error-prone tasks. Throughout the thesis, I dually pursue theoretical aspects of the design process as well as issues of support environments. Initially, the literature is reviewed for weak aspects of the design process, i.e., parts not supported by standard software tools. Our focus turns on the 'breakdowns', which correspond to cognitive difficulties faced by designers. Interestingly enough, another research strand emphasizes on the recovery from such difficulties seeing it as facilitation of problem re-framing and generation of ideas. It is suggested that in order to facilitate that type of recovery transition, the decision making process should be assisted through the sharing of expertise among design stakeholders.

I then perform two studies which enable me to better understand design practice. Work on the SEDRES project included a field study of design practice in the aerospace industry. A host of information on designers at work using software tools is gathered, of both theoretical and practical value and many of the raised research issues were addressed. On a separate instance, an exploratory study was performed in order to establish a relationship between breakdown-related behaviour and decision making. In a laboratory setting, an analysis of the verbal protocols of five designers showed a high correlation of these two types of behaviour and yielded a prototype model of decision making in software design as well as a number of ideas on how to facilitate transition processes of the forementioned kind.

I am now in position to form specific hypotheses about the usefulness of design rationale in this new role. In two controlled experiments, I test certain representational and structural aspects of design rationale for the purpose of transferring expertise. In the first experiment, certain merits of notation and nature of rationale are attributed to particular linguistic features of the formalism that is used. Those results are used in the formation of a novel theory of comprehension of design rationale by novice analysts. Secondly, in a quasi experimental setting, the re-use of previous designs under different conditions of given information and its representation is investigated. It is advocated that it is the very nature of argumentation that it is useful in putting design diagrams into context with the original process of creating them as it puts the reader into a 'virtual discussion' with the design material and brings direct benefits to them. These two experiments set up the basis for the adoption of design rationale as an essential supplement to conventional design documentation.

More work is needed in order to establish an issue-based software process as a commodity of practice. In theoretical terms, we need to provide formalisms that are tailor-made to tasks, domains and problems, in general. In practical terms we need to put formalism and associated tools into industry and make them a cultural part of the software design community.
Acknowledgements

Many thanks go to

- The State Scholarships Foundation of Greece (IKY) for funding this research and for being prompt and helpful, throughout. Their contribution is gratefully appreciated.

- My supervisor, Prof E A Edmonds, who through the SEDRES project gave me the opportunity to gain valuable ‘real world’ experience.

- Several people who raised ideas and provided useful feedback and stimulating discussion at different stages, including Ghada Kadoda, Pete Hornsby, Alan Blackwell, Simon Buckingham Shum, Francoise Detienne, Trevor Collins, Paul Mulholland, and especially Tony Clarke, Thomas Green and Raquel Navarro-Pneto.

- My family, Takes, Elene and Tasos for their love and support.

- Maria, Zeta, Giorgo and Katerina for being such good friends.
List of contents

PART A: Establishing the Background

CHAPTER 1: INTRODUCTION
- 1.1 Field of study: Empirical investigations of software design ......................................... 2
- 1.2 Reviewing previous work: Breakdowns and knowledgeable recoveries ......................... 4
- 1.3 Advancing to present research: Study of tacit design contexts ..................................... 4
- 1.4 Aims and approach ........................................................................................................ 5
- 1.5 Contribution of thesis .................................................................................................... 5
- 1.6 Organisation of thesis report ........................................................................................ 7

CHAPTER 2: A human process-centred view of software design
- 2.1 Overview of literature survey ...................................................................................... 11
- 2.2 Research background ................................................................................................. 11
- 2.3 Establishing a context: The software design factors .................................................... 12
- 2.4 Empirical studies of software design .......................................................................... 18
  - 2.4.1 By dynamics ............................................................................................................ 18
    - 2.4.1.1 Individual level .................................................................................................. 18
    - 2.4.1.2 Social/team level ............................................................................................... 19
    - 2.4.1.3 Organisational level ......................................................................................... 19
  - 2.4.2 Other studies organised by paradigm ..................................................................... 20
    - 2.4.2.1 Participatory design .......................................................................................... 20
    - 2.4.2.2 Object-oriented design ...................................................................................... 20
  - 2.4.3 Summary of findings .............................................................................................. 21
  - 2.4.4 Classifying processes ............................................................................................. 21
  - 2.4.5 A design process framework .................................................................................. 31
- 2.5 Software tools for software design .............................................................................. 34
  - 2.5.1 Choice of tools ....................................................................................................... 34
  - 2.5.2 Description of tools ............................................................................................... 35
- 2.6 The software design support framework (SDSF): How well tools support design activities ................................................ 38
  - 2.6.1 Aims of the SDSF methodology .......................................................................... 38
  - 2.6.2 Outline .................................................................................................................. 38
  - 2.6.3 Analytical form ...................................................................................................... 39
  - 2.6.4 Synoptic form ....................................................................................................... 43
- 2.7 Main observations ........................................................................................................ 46
PART B: Understanding and Modelling Design Practice

CHAPTER 3: THE SEDRES PROJECT

3.1 INTRODUCTION ......................................................................................... 63
  3.1.1 PROJECT SUMMARY ........................................................................... 63
    3.1.1.1 OVERVIEW .................................................................................. 63
    3.1.1.2 AIMS ......................................................................................... 63
    3.1.1.3 OBJECTIVES .............................................................................. 63
    3.1.1.4 THE PROBLEM CONTEXT ............................................................. 64
    3.1.1.5 RELATED WORK ......................................................................... 64
    3.1.1.6 FURTHER INFORMATION ............................................................... 65
  3.1.2 THE ROLE OF SEDRES IN THE THESIS ........................................... 65

3.2 WORK PACKAGE 3 (WP3): 'DEFINE PROCESS' ........................................... 66
  3.2.1 OBJECTIVE ....................................................................................... 66
  3.2.2 APPROACH ....................................................................................... 66
  3.2.3 TASKS 3.1, 3.3 .................................................................................. 66
    3.2.3.1 OBJECTIVES ............................................................................. 66
    3.2.3.2 APPROACH ................................................................................ 66
    3.2.3.3 DESIGNERS INTERVIEWS .......................................................... 68
    3.2.3.4 SURVEYS: DISCUSSIONS AND FEEDBACK FORMS .................. 69
    3.2.3.5 DEVELOPMENT OF A HUMAN-ORIENTED HYBRID NOTATION FOR THE SYSTEMS ENGINEERING PROCESS ..................................... 70
  3.2.3.6 THE SEDRES GENERIC PROCESS MODEL (SGPM) ....................... 74
  3.2.3.7 HARMONISING PARTNERS MODELS W.R.T. THE GPM .............. 78
  3.2.3.8 CONCLUSIONS AND RELATIONSHIPS TO SUBSEQUENT TASKS (T3.2, T3.3) ................................................................. 82
Part C: Experimental work - Expertise transfer using design rationale

CHAPTER 5: DESIGN RATIONALE TOOLS

5.1 INTRODUCTION

5.2 DESIGN RATIONALE

5.2.1 DEFINITION

5.2.2 PARADIGMS AND TOOLS

5.2.2.1 IBIS AND RIBIS - QUESTMAP - DRAMA, PHI AND PHIDIAS - JANUS

5.2.2.2 QOC AND RELATED TOOLS

5.2.2.3 Potts & Bruns, DRL and SIBYL, QAR AND DEBATE BROWSER

5.2.3 DR IN SOFTWARE DESIGN

5.3 THE LOUIS SYSTEM

5.3.1 FEATURES

5.3.2 DEVELOPMENT PLATFORM

5.3.2.1 TCU/TK

5.3.3 ARCHITECTURE

5.3.4 USAGE SCENARIOS

5.3.5 USER TRIALS

5.3.6 PLANNING WITH LOUIS

5.4 DESIGN RATIONALE AS A FACILITATOR OF EXPERTISE TRANSFER

5.4.1 WHAT DOES EXPERTISE CONSTITUTE OF?

5.4.2 REPRESENTING EXPERTISE FRAGMENTS USING DR

5.5 CHAPTER SUMMARY

CHAPTER 6: INVESTIGATING ISSUES OF STRUCTURE AND REPRESENTATION IN THE RE-USE OF DESIGN RATIONALE

6.1 INTRODUCTION

6.2 HYPOTHESES

6.2.1 DR REPRESENTATION

6.2.2 DR STRUCTURE

6.3 METHOD

6.3.1 MATERIALS

6.3.2 SUBJECTS

6.3.3 DESIGN

6.3.4 PROCEDURE

6.3.5 SCORING CONVENTIONS
REFERENCES................................................................. ................................. 211

APPENDIX A: Documents related to chapter 3 (The SEDRES project)

SEDRES GLOSSARY................................................................. .......................... 218
STRUCTURED INTERVIEW................................................................. 224
FEEDBACK FORM................................................................. 235
THE GENERIC PROCESS MODEL NOTATION TUTORIAL................................. 237
COMPANIES PROCESS MODELS................................................................. 246
MAJOR EVALUATION DATA COLLECTION FORMS........................................... 251
TRANSFORM REPORT EXAMPLES................................................................. 256

APPENDIX B: Documents related to chapter 4 (The PA study)

THINK-ALOUD TRAINING SESSION ................................................................. 260
PROBLEM STATEMENT................................................................. 261
DEBRIEFING SESSION................................................................. 262

APPENDIX C: Documents related to chapter 6 (Experiment #1)

A TUTORIAL IN DESIGN RATIONALE USING QOC............................................. 264
RANDOMISATION TABLE................................................................. 271
EXPERIMENTAL TASKS................................................................. 272
STIMULUS MATERIAL................................................................. 281
QUESTIONNAIRE................................................................. 289

APPENDIX D: Documents related to chapter 7 (Experiment #2)

DESIGN DOCUMENTATION................................................................. 302
DESIGN RATIONALE DOCUMENTATION................................................................. 304
List of figures

Chapter 1

Figure 1a: The three aspects of process improvement in software design

Chapter 2

Figure 2a: Giddings's software classification
Figure 2b-a: Design process framework – part 1
Figure 2b-b: Design process framework – part 2
Figure 2b-c: Design process framework – part 3
Figure 2c-a: Analytical form of the Software Design Support Framework (SDSF) – part 1
Figure 2c-b: Analytical form of the Software Design Support Framework (SDSF) – part 2
Figure 2c-c: Analytical form of the Software Design Support Framework (SDSF) – part 3
Figure 2d-a: Synoptic form of the Software Design Support Framework (SDSF) – part 1
Figure 2d-b: Synoptic form of the Software Design Support Framework (SDSF) – part 2
Figure 2d-c: Synoptic form of the Software Design Support Framework (SDSF) – part 3
Figure 2e: The new version of SDSF
Figure 2f: Winograd and Flores's view of breakdowns as conceptualisations
Figure 2g: Fischer's view of breakdowns in design problem solving
Figure 2h: A collective view of breakdown theories
## Chapter 3

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3a</td>
<td>Proposed view of improvement of the concurrent engineering process</td>
<td>69</td>
</tr>
<tr>
<td>3b</td>
<td>The SEDRES Generic Process Model notation</td>
<td>73</td>
</tr>
<tr>
<td>3c</td>
<td>Genenc systems engineering lifecycle</td>
<td>75</td>
</tr>
<tr>
<td>3d</td>
<td>Systems engineering lifecycle including the software process</td>
<td>75</td>
</tr>
<tr>
<td>3e</td>
<td>Genenc systems engineering process</td>
<td>76</td>
</tr>
<tr>
<td>3f</td>
<td>Placement of Use Processes In the context of the SGPM</td>
<td>77</td>
</tr>
<tr>
<td>3g</td>
<td>The Use Scenario 1 process (functional perspective) represented using the SGPM notation</td>
<td>84</td>
</tr>
<tr>
<td>3h</td>
<td>The Use Scenario 1 process – timeline view</td>
<td>84</td>
</tr>
<tr>
<td>3i</td>
<td>Identification of the role of humans and tools in the US1 process</td>
<td>85</td>
</tr>
<tr>
<td>3j</td>
<td>Overview of Workpackage 6 activities</td>
<td>86</td>
</tr>
<tr>
<td>3k</td>
<td>Checklist for performing an on-site observation</td>
<td>89</td>
</tr>
<tr>
<td>3l</td>
<td>Overview of data collection activities within the context of an exchange</td>
<td>90</td>
</tr>
<tr>
<td>3m</td>
<td>A NU*DIST screen shot</td>
<td>91</td>
</tr>
<tr>
<td>3n</td>
<td>Number of text units referring to each usability requirement for each exchange group</td>
<td>94</td>
</tr>
<tr>
<td>3o</td>
<td>Number of text units for each usability requirement for each tool as export tool</td>
<td>94</td>
</tr>
<tr>
<td>3p</td>
<td>Number of text units for each usability requirement for each tool as import tool</td>
<td>95</td>
</tr>
<tr>
<td>3q</td>
<td>Information restrictions posed by methods on observation data</td>
<td>96</td>
</tr>
<tr>
<td>3r</td>
<td>Summary of snag-related nodes</td>
<td>97</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>4a</td>
<td>S3's breakdown episode BD#1</td>
<td>108</td>
</tr>
<tr>
<td>4b</td>
<td>S3's breakdown episode BD#2</td>
<td>108</td>
</tr>
<tr>
<td>4c</td>
<td>S3's breakdown episode BD#3</td>
<td>109</td>
</tr>
<tr>
<td>4d</td>
<td>S3's breakdown episode BD#4</td>
<td>110</td>
</tr>
<tr>
<td>4e</td>
<td>S3's breakdown episode BD#5</td>
<td>110</td>
</tr>
<tr>
<td>4f</td>
<td>S3's decision making episode DM#1</td>
<td>111</td>
</tr>
<tr>
<td>4g</td>
<td>S3's decision making episode DM#2</td>
<td>112</td>
</tr>
<tr>
<td>4h</td>
<td>S3's decision making episode DM#3</td>
<td>113</td>
</tr>
<tr>
<td>4i</td>
<td>S3's decision making episode DM#4</td>
<td>114</td>
</tr>
<tr>
<td>4j</td>
<td>S4's breakdown episode BD#2</td>
<td>117</td>
</tr>
<tr>
<td>4k</td>
<td>S4's breakdown episode BD#3</td>
<td>118</td>
</tr>
<tr>
<td>4l</td>
<td>S4's breakdown episodes BD#4-5</td>
<td>121</td>
</tr>
<tr>
<td>4m</td>
<td>S4's breakdown episode BD#4a</td>
<td>122</td>
</tr>
<tr>
<td>4n</td>
<td>S4's breakdown episode BD#4a revisited</td>
<td>123</td>
</tr>
<tr>
<td>4o</td>
<td>S4's breakdown episode BD#4 revisited</td>
<td>124</td>
</tr>
<tr>
<td>4p</td>
<td>S4's decision making episode DM#1</td>
<td>127</td>
</tr>
<tr>
<td>4q</td>
<td>S4's decision making episode DM#2</td>
<td>128</td>
</tr>
<tr>
<td>4r</td>
<td>S4's decision making episode DM#3</td>
<td>129</td>
</tr>
<tr>
<td>4s</td>
<td>S4's decision making episode DM#4</td>
<td>130</td>
</tr>
<tr>
<td>4t</td>
<td>S4's decision making episode DM#6</td>
<td>131</td>
</tr>
<tr>
<td>4u</td>
<td>S4's decision making episode DM#8</td>
<td>132</td>
</tr>
<tr>
<td>4v</td>
<td>S4's decision making episode DM#12</td>
<td>133</td>
</tr>
<tr>
<td>4w</td>
<td>Overview of coincidence of breakdowns to decision making over process stages for S4</td>
<td>133</td>
</tr>
</tbody>
</table>
### Chapter 5

<table>
<thead>
<tr>
<th>Figure 5a: IBIS generic structure</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 5b: An example of the application of IBIS to kitchen design</td>
<td>140</td>
</tr>
<tr>
<td>Figure 5c: A screen shot from QuestMap</td>
<td>141</td>
</tr>
<tr>
<td>Figure 5d: The DRAMA tool</td>
<td>142</td>
</tr>
<tr>
<td>Figure 5e: A quasi-hierarchical structure of subissues as the model behind PHI</td>
<td>143</td>
</tr>
<tr>
<td>Figure 5f: Application of PHI to the same problem as in figure (5b)</td>
<td>144</td>
</tr>
<tr>
<td>Figure 5g: QOC generic syntax and grammar</td>
<td>144</td>
</tr>
<tr>
<td>Figure 5h: The Potts &amp; Bruns argumentation model</td>
<td>146</td>
</tr>
<tr>
<td>Figure 5i: Generic vocabulary and structure of DRL</td>
<td>147</td>
</tr>
<tr>
<td>Figure 5j: The QAR method</td>
<td>148</td>
</tr>
<tr>
<td>Figure 5k: A Debate Browser screen shot</td>
<td>149</td>
</tr>
<tr>
<td>Figure 5l: LOUIS use scenario 1 - creation of a new issue and its associated rationale</td>
<td>152</td>
</tr>
<tr>
<td>Figure 5m: LOUIS use scenario - exploring the design space in two levels: issue clustering and design rationale management</td>
<td>153</td>
</tr>
<tr>
<td>Figure 5n: LOUIS use scenario 3 - DR as a supplement of the design. Four types of design knowledge combined: problem statement, (Problem text editor), design product representation (OODesigner CASE tool), design rationale (LOUIS – DR Viewer), and simulation/annotation (WhiteBoard)</td>
<td>153</td>
</tr>
</tbody>
</table>

### Chapter 6

<table>
<thead>
<tr>
<th>Figure 6a: Different QOC representations</th>
<th>161</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 6b: Overview of an experimental design rationale re-use task</td>
<td>163</td>
</tr>
<tr>
<td>Figure 6c: Boxplots of the distributions of correct answers over the 3 types of representation</td>
<td>171</td>
</tr>
<tr>
<td>Figure 6d: Scatterplot of the correlation between two distributions of response time: observed Vs normalised</td>
<td>174</td>
</tr>
<tr>
<td>Figure 6e: Bar charts of means of preference ratings for the three types of representation</td>
<td>187</td>
</tr>
<tr>
<td>Figure 6f: Pie chart showing proportions of overall preferences of notational form</td>
<td>187</td>
</tr>
</tbody>
</table>
List of tables

Chapter 1

Chapter 2

| Table 2a: List of software tools and their associated domains | 34 |
| Table 2b: Explanation of the symbolisms used on SDSF | 39 |

Chapter 3

| Table 3a: SEDRES process cross-reference table | 79 |
| Table 3b: Number of text units referring to each usability requirement for each exchange group | 93 |

Chapter 4

Chapter 5

| Table 5a: Some contrasting features between well- and ill-structured problems [S18] | 138 |
Chapter 6

<table>
<thead>
<tr>
<th>Table 6a: Demographics of the experimental sample</th>
<th>165</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 6b: List of abbreviations</td>
<td>169</td>
</tr>
<tr>
<td>Table 6c: Main descriptive measures of correct answers on representation</td>
<td>169</td>
</tr>
<tr>
<td>Table 6d: Main descriptive measures of correct answers on representation over task types</td>
<td>170</td>
</tr>
<tr>
<td>Table 6e: Testing the normality of the distribution of the percentage of correct answers over DR representation conditions</td>
<td>171</td>
</tr>
<tr>
<td>Table 6f: Testing the normality of the distribution of the percentage of correct answers over DR representation conditions for different types of tasks</td>
<td>172</td>
</tr>
<tr>
<td>Table 6g: Results of ANOVA test for overall effects of notation to correct answers</td>
<td>172</td>
</tr>
<tr>
<td>Table 6h: Main descriptive measures of response time on representation</td>
<td>173</td>
</tr>
<tr>
<td>Table 6i: Main descriptive measures of response time on representation over task type</td>
<td>173</td>
</tr>
<tr>
<td>Table 6j: Testing the normality of the distribution of response time over DR representation</td>
<td>174</td>
</tr>
<tr>
<td>Tables 6k: Sphericity and subsequent ANOVA tests for response time (overall)</td>
<td>175</td>
</tr>
<tr>
<td>Table 6l: Paired samples t-tests for response time (overall)</td>
<td>175</td>
</tr>
<tr>
<td>Tables 6m: Sphericity and subsequent ANOVA tests for response time over syntax tasks</td>
<td>176</td>
</tr>
<tr>
<td>Table 6n: Paired samples t-tests for response time over syntax tasks</td>
<td>176</td>
</tr>
<tr>
<td>Tables 6o: Sphericity and subsequent ANOVA tests for response time over semantics tasks</td>
<td>177</td>
</tr>
<tr>
<td>Table 6p: Paired samples t-tests for response time over semantics tasks</td>
<td>178</td>
</tr>
<tr>
<td>Tables 6q: Sphericity and subsequent ANOVA tests for response time over context tasks</td>
<td>179</td>
</tr>
<tr>
<td>Table 6r: Main descriptive measures of percentage of correct answers over argumentative structure (overall)</td>
<td>180</td>
</tr>
<tr>
<td>Table 6s: Main descriptive measures of percentage of correct answers over argumentative structure over task type</td>
<td>180</td>
</tr>
<tr>
<td>Table 6t: Testing the normality of the distribution of correct answers over DR structure conditions (overall)</td>
<td>181</td>
</tr>
</tbody>
</table>

(continuing on next page)
Table 6u: Testing the normality of the distribution of correct answers over DR structure conditions for different task types 181

Table 6v: Results of independent samples t-test for the overall effect of structure on correct answers 182

Table 6w: Results of independent samples t-test for the effect of structure on correct answers over task type 182

Table 6x: Main descriptive measures of response time over argumentative structure (overall) 183

Table 6y: Main descriptive measures of response time over argumentative structure over task type 183

Table 6z: Testing the normality of the distribution of response time over DR structure conditions (overall) 184

Table 6z-a: Testing the normality of the distribution of response time over DR structure conditions for different task types 184

Table 6z-b: Results of independent samples t-test for the overall effect of structure on response time 185

Table 6z-c: Results of independent samples t-test for the effect of structure on correct answers over task type 185

Table 6z-d: Main descriptive measures of ratings of preference for DR representations 186

Chapter 7

Table 7a: Rhetorical elements included in the experimental documents. Elements towards the upper part of the table tend to be more static, as parts of the final product and its evaluation, whereas elements towards the lower end of the table tend to be more dynamic in that they reflect problems/issues raised during the design process 197

Chapter 8
PART A: Establishing the Background

CHAPTER 1

INTRODUCTION

Table of contents

1.1 Field of study: Empirical investigations of software design ... 2
1.2 Reviewing previous work: breakdowns and knowledgeable recoveries ... 4
1.3 Advancing to present research study of tacit design contexts ... 4
1.4 Aims and approach ... 5
1.5 Contribution of thesis ... 5
1.6 Organisation of thesis report ... 7
1.1 Field of study: empirical investigations of software design

The discrepancy between what people say or think they do and what they actually do is well known in Psychology. In the context of software design, prescriptive methods suggest clearly defined stages of the development process albeit differing in their philosophy on what these stages are. However, empirical research on designers reveals a different view: opportunistic behaviour and interchange between knowledge domains with expertise and familiarity with the problem domain playing a large role.

In particular, this work is concerned with the upstream portion of software design, i.e. the transition from a problem statement (user requirements) to a solution (high-level design) and the elaboration of this solution up to a level where it can be easily computer-programmed. Recently there has been a tendency to view software design more as a creative activity without however, underestimating the engineering part of it. Therefore, this study is trying to discover things like what exactly happens during high-level software design, which ingredients in the process give the design its creative flavour and which (if any) are the aspects of designing software that tools fail to support.

There are indications in the literature that there is space for improvement in the practice of software design. This improvement is meant at least in terms of process optimisation i.e. achieving a solution faster, efficiency i.e. analyzing the problem thoroughly, validation i.e. assuring that the product is throughout in line with the current goals, and creativity, i.e. producing solutions based on novel aspects of the problem space. Thus, the overall aim of this research is to provide information that will lead to the improvement of software productivity in terms of quality of software and designers' efficiency.

Although in the history of computers and software design/development many big events have taken place which have provided assistance to override the inherent complexity of this task, still some of the original problems have not been solved. Let us be more specific and see how different research progresses have contributed to the development of software.

Software engineering is an engineering discipline concerned with the practical problems of developing large software systems. It therefore brings along with it a set of techniques, methods, process models and tools that ensure that software has the following attributes: maintainability, reliability, efficiency and a proper user interface [85]. Some of the general models or paradigms of the software process are the waterfall model, the prototyping model and formal transformation. The latter involves developing a formal specification of the software system and transforming this specification, using correctness-preserving transformations, to a program [85, 65]. Another relevant discipline is HCI (Human-Computer...
Interaction) whose software development aspect deals with designing/developing user interfaces i.e. the part of computer software that handles the dialogue with the user.

However, none of the above offer any clues on either for example how to come up with the original system architecture or how to decompose the problem best. These are only two examples of a series of processes that take place in software design which more or less leave practitioners and researchers in the dark in terms of how exactly they happen, what kind of low-level (cognitive) operations they involve, and how they could be supported by techniques, tools, methods etc.. Therefore, there is on the one hand a lack of knowledge for some aspects of software design, and consequently on the other hand a space for improvement of design practice towards a few perspectives. As I will try to show in my research, by shedding some light on these questions we may find better optimised, more efficient and more creative ways to design software.

A possible route for this improvement is from process to practice (tools) to theoretical models like methods etc. or even as shown in figure (1a):

![Figure 1a: The three aspects of process improvement in software design](image)

So the approach that is taken is that software designers are humans and therefore in order to understand their behaviour they will have to be treated as every other prospective user of a computer system and the same techniques will have to be used in order to get to know their requirements and needs.

The reason for this insistence on real design practice is the feel that the vast majority of software is domain- and context-dependent. That happens because no two computer systems are alike; the users are always different and the work environments are always different [14]. This realistic character, in turn, necessitates the use of a process-based approach in trying to understand and model software design. Cohill [14] offers a very good justification for that:
"...yet as every system developer knows, the goals of a project change constantly as development proceeds." ...the introduction of a new system changes the operating environment on which the original goals were based. So the operating environment for which the system was designed does not exist when it is delivered. The act of designing changes the world and only a design model that incorporates feedback as a basic tenet will be robust. Therefore a method is static, producing the same results each time it is used, while a process is a dynamic entity.

The main assumption made in this work is that software design is a problem solving activity. This is an adoption of the view of many scientists such as Rittel [77], and Nii [66]. They go further and advocate that the problems in software design are ill-structured (this issue is covered in detail in section 5.2.2) having among others the characteristic that there is no pre-defined path to get to the solution [40]. This feature gives software design its artistic flavour and allows for creativity, because designers are free to improvise. That is also where expertise plays a large role. It also makes this study very interesting as on the one hand many people e.g. Glass [33] and also others have reported similar artistic influence but the actual progress of research in this direction is relatively small. On the other hand, computers should be used more and more as intellectual assistants in complicated human tasks, and not merely as repetitive dumb machines.

1.2 Reviewing previous work: breakdowns and knowledgeable recoveries

In studying design behaviour, the starting point is breakdowns, one of the key processes in software design. Breakdowns mainly relate to the difficulties [35] humans find in decomposing and merging complex problem structures. From a more positive perspective, breakdown recovery can be seen as a process that facilitates discovery of knowledge and generation of ideas [25] Apart from their high correlation with other cognitive activities, breakdowns are important because they can be predictors of errors in the design of systems. These design errors can lead to problems in the use of resulting systems. Getting to know the error-prone properties that are inherent in a problem is a start to improving the process of solving it.

1.3 Advancing to present research: study of tacit design contexts

It would be interesting from a theoretical perspective to see how types of reasoning relate to breakdowns. In particular, can effective reasoning in the form of decision making or argumentative reasoning systematically affect the occurrence of breakdowns? What about human errors at several phases in the process? This leads us to focus more onto Rittel's work on design rationale and issue-based approaches to software design [77].
From a practical perspective, are there any features in a design rationale language that really assist in design? What can we learn about how to design better tools?

In other words, breakdowns and related processes are used as a platform of looking at the design process and design rationale is used as a platform on which to study design reasoning. Using these 'platforms', a new experimental basis for the study of software design is formed, as I can perform manipulations on designers' reasoning and measure their effect on the design process and consequently, on the quality of the outcome.

1.4 Aims and approach

The objective of the thesis is to investigate the effectiveness of argumentation-based methods in supporting the process of software design, focusing in particular on circumstances where breakdown occurs in the design process. In light of the experimental basis mentioned above, two paths are laid out in realising this objective. (a) the deeper understanding of the process of high-level software design and (b) the provision of information that directly involves improvement of the design process

The approach that is followed to meet these objectives and to answer the questions put in section 1.3, is as follows: a protocol analysis study is performed to find out how breakdown occurrence and decision making behaviour relate I thereby get an idea of potential usefulness of design rationale in coping with breakdowns and related incidents and produce a set of 'guidelines for breakdown management'. I then create breakdown-prone environments and check attributes of design rationale in facilitating a range of breakdown recoveries like transfer of domain knowledge and generation of ideas.

It is a question of communicating among designers 'potential' design errors and resolutions; that is how expertise transfer is facilitated. It is then suggested that plain expertise transfer - even coming out of a near perfect design - by itself would not suffice as it is out of context. An error (breakdown) is the mechanism to place expertise in the right context - in fact, design rationale is particularly suited to that process.

1.5 Contribution of thesis

Several results from this report constitute novel contribution to knowledge in the subject of Design Rationale as well as to the field of the Psychology of Programming and the pursuit of a better understanding and description of the human processes involved in software construction. In this section, a brief account of those results is given.
With regards to the better understanding and modelling of the design process, there are two types of modelling activities that produced interesting results. One stream of such results comes from the SEDRES project, where the common systems engineering process that underlies the products of the European aerospace companies is modelled. To manage such a task I initially define the levels of abstraction on which the processes manifest themselves, then combine standard notations with the notion of the hi-graph to provide for encapsulation on composite processes and finally I allocate special annotations to reflect the role of humans and tools in the process. The result is the SEDRES General Process Model (SGPM).

The other stream of 'modelling' results comes from the protocol analysis study, in which I explore the correlation of breakdowns and decision making from the perspective of two characteristic user profiles. Modelling work reveals correlation of different types of breakdowns – namely difficulties and conceptualisations – with respect to the rest of the design behaviour and differentiating according to designers’ profile I outline the decision types for both profiles and how those differ and note a set of cases in which these two processes (breakdowns and decision making) are related in (a) coincidence in general, (b) patterns of causal effects. Those results underpin my pursuit of knowledgeable recovery and the potential transformation of difficulties to conceptualisations through knowledge acquisition and transfer of expertise.

In terms of experimental work, experiment #1 was the main provider of concrete data, considering that experiment #2 was a study with more of a confirmatory or speculative character. Experiment #1 investigated the effect of the representation used by different design rationale formalisms as well as the structure of the associated argumentation on the comprehension of design rationale by novice analysts. Results include: (a) a high degree of comprehension of design rationale and engagement in the underlying argumentation (b) better understanding of graphical design rationale material over its tabular and narrative counterparts, (c) significantly quicker comprehension of tabular design rationale fragments than their graphical and narrative counterparts — with the exception of nested fragments where graphs did better (d) overall preference of graphs over tables and narratives, (e) options-to-criteria structure preferred to criteria-to-options one by novices.

Experiment #2 confirmed prior expectations that it is feasible — though probably not entirely sufficient — to facilitate effects of skill transfer through re-use of design rationale over certain tasks that are error-prone.
1.6 Organisation of thesis report

The thesis is divided in four parts. Part A establishes the background upon the current literature.

Chapter 1 places the thesis in context with current research on empirical studies of software design and outlines and justifies its purpose and approach. By doing that it orientates the reader through the main issues that are dealt with in the intermediate chapters.

Chapter 2 provides an overview of the background literature presenting other relevant pieces of work and describes the main bibliographic sources. It starts off by reviewing previous work on empirically modelling the software design process and then reviews a set of software tools that assist design, setting the scene for SDSF (Software Design Support framework). SDSF is a summarizing framework of the type of support that current tools offer for a range of cognitive processes that source in different design situations. Using SDSF, a set of interesting design activities are identified - interesting in that they do not receive adequate support from software tools. These activities are centred around ‘breakdowns’ in designers’ behaviour. The chapter finishes with a separate survey of the literature on breakdowns themselves.

Part B describes two studies which enabled us to gain a better understanding of the software design process in different environments and contexts.

Chapter 3 describes the author’s experience through his involvement in the SEDRES project - an ESPRIT project involving systems engineering practice in the aerospace industry. That experience took the form of a field study as it enabled direct access to design practice in the ‘real world’. The chapter is arranged according to the project work packages. It first describes large scale process modelling work and then outlines evaluation work. It finishes by summarizing the main issues that were addressed which relate to this thesis.

Chapter 4 describes a protocol analysis study that explored decision making and breakdown behaviour as well as their relationship. This time the setting was based in a laboratory and participants included research students and software engineers. The chapter starts off with setting out the goals and methodological approach of the study. Then the results from analysis of two chosen protocols are presented. At the end of the chapter, all findings are summarised and the scene is set for a design approach based on argumentative reasoning.
Part C presents software development and experimental work.

Chapter 5 surveys design rationale tools. Such tools provide the basis for knowledge transfer among design stakeholders. I initially present design rationale research analyzing the founding theories and software packages that come with them. Then the LOUIS system is described, an experimental design rationale tool developed by the author, and finally the components of expertise are investigated, as well as how design rationale research could be extended to accommodate its capture and sharing among designers.

Chapter 6 describes a controlled experiment which investigated issues of design rationale structure and representation by comparing a set of alternative 'standard' approaches. Initially, I set out the hypotheses that underlie the three types of representation (narrative, tabular and graphical) and the two types of argumentative structure (options-based, criteria-based) that are compared. Then, the experimental method is laid out and finally the results are presented and interpreted.

Chapter 7 presents a quasi experiment which focuses on the active problem-solving part of design re-use. The usefulness of design rationale as complementary documentation is put into the test and for that certain tasks are constructed. As with the previous experiment, I first set out the background of the study and the goals and hypotheses and then describe the method that was used. The chapter ends up by presenting the main results which include interesting skill transfer phenomena.

Part D concludes the thesis.

Chapter 8 includes a collective discussion of all the work in the thesis. Through that discussion, the thesis is evaluated as to what extent the work that it describes has met the objectives of the thesis. Certain recommendations are then made on how one could elaborate on the existing piece of work and a research agenda for future work is set.
CHAPTER 2

A HUMAN PROCESS-CENTRED VIEW OF SOFTWARE DESIGN

Table of contents

2.1 OVERVIEW OF LITERATURE SURVEY .................................................. 11
2.2 RESEARCH BACKGROUND .............................................................. 11
2.3 ESTABLISHING A CONTEXT: THE SOFTWARE DESIGN FACTORS .......... 12
2.4 EMPIRICAL STUDIES OF SOFTWARE DESIGN .................................. 18
  2.4.1 BY DYNAMICS ................................................................. 18
    2.4.1.1 INDIVIDUAL LEVEL ................................................. 18
    2.4.1.2 SOCIAL/TEAM LEVEL .............................................. 19
    2.4.1.3 ORGANISATIONAL LEVEL ......................................... 19
  2.4.2 OTHER STUDIES ORGANISED BY PARADIGM .............................. 20
    2.4.2.1 PARTICIPATORY DESIGN .......................................... 20
    2.4.2.2 OBJECT-ORIENTED DESIGN .................................... 20
  2.4.3 SUMMARY OF FINDINGS ..................................................... 21
  2.4.4 CLASSIFYING PROCESSES ................................................... 21
    2.4.5 A DESIGN PROCESS FRAMEWORK ..................................... 31
2.5 SOFTWARE TOOLS FOR SOFTWARE DESIGN .................................... 34
  2.5.1 CHOICE OF TOOLS .................................................................. 34
  2.5.2 DESCRIPTION OF TOOLS ..................................................... 35
2.6 THE SOFTWARE DESIGN SUPPORT FRAMEWORK (SDSF): HOW WELL TOOLS SUPPORT DESIGN ACTIVITIES ................................................................. 38
  2.6.1 AIMS OF THE SDSF METHODOLOGY ................................... 38
  2.6.2 OUTLINE .............................................................................. 38
  2.6.3 ANALYTICAL FORM ............................................................ 39
  2.6.4 SYNOPTIC FORM .................................................................. 43
2.7 MAIN OBSERVATIONS ................................................................. 46
2.8 DISCUSSION .................................................................................... 46
  2.8.1 SDSF TOOLS ....................................................................... 48
  2.8.2 SDSF AS A SOFTWARE PROCESS MODEL? ............................... 48
2.9 SDSF REFINEMENT ................................................................. 49
  2.9.1 REFINEMENT OF TOOL SECTION ....................................... 49
  2.9.2 THEORETICAL FUTURE WORK ........................................... 50
2.10 Breakdowns in the Design and Use of Software Systems ........................................ 52
  2.10.1 Breakdowns as Difficulties in Designing Software ........................................ 52
  2.10.2 Breakdowns as Conceptualisations .............................................................. 53
  2.10.3 Breakdowns as Mismatches of Mental Models ............................................... 56
  2.10.4 Discussion .................................................................................................. 57

2.11 The Prospect of Theories of Argumentative Reasoning with Process Support 58

2.12 Chapter Summary ............................................................................................ 60
2.1 Overview of literature survey

In this chapter, I investigate the current status of support that software tools provide to designers from a perspective of actual practice. It is also speculated upon what could be done to improve that support. Initially, a work context is established by categorising design activities in rigid classes, comprising a context schema. Its purpose is to establish a clear research scope making the work comparable and communicative to that of others. I then survey and critically appraise the literature on empirical studies of software design and classify them according to the newly defined schema. That is followed by a survey of support tools. Consequently, the quality of tool support to design activities is investigated by cross-referencing the results of the previous two surveys. The result is the Software Design Support Framework (SDSF). By studying this framework I make some remarks which drive the study to breakdown design behaviour. A comprehensive review of the literature on breakdowns and all related activities is performed, bringing different studies together and defining an own view on the matter. Lastly, personal thoughts are outlined on how argumentative reasoning could provide an insight into design cognitive processes and also support in their facilitation, thereby preparing the ground for the second part of this work which deals with aspects of design rationale. A summary of the main literature survey is included at the end of the chapter.

2.2 Research background

The initial research incentive was to study phenomena of methodology deviation in software design. The questions that consequently led the initial literature survey, i.e. "what do software designers do?"", "why they work that way?" and "what tools are there to support them?" have yielded three points of emphasis in this research: (i) a better understanding of the software design process (stream #1), (b) the description of this process from a more formal point of view, i.e. how formal techniques, methodologies, generic process models etc. perceive this process and model it for the sake of disciplined and tidier design (stream #2) and (c) a close examination of the current tool support for it (stream #3). Similarly, information gathering is organized in these three subject categories, plus a fourth one which includes general type of literature for the topic, e.g. material concerned with the usability of the potential computer-supported design.

Stream #1 includes a series of studies of software design behaviour. There are mainly two types: (a) observational or field studies where experimenters have been merely observing software designers at their place of work, and (b) empirical studies where designers are brought into laboratory settings, presented with a set of tasks which they are supposed to carry out. The sessions in both types of studies are recorded and then the resulting protocols are analysed. The conveyed information is very useful because it provides a very realistic source of design practice. The studies vary in context quite a lot: there are situations where a designer is given a requirements specification document, i.e. ideally models of the users and
their tasks and is expected to come up with a design. Some other situations include a group of designers co-operating for the production of a design, some others include dialogues between a designer and a prospective user, and so forth. However, it must be said that all of them refer to high-level design of problems of realistic complexity. Typical references: [19, 35, 95].

Stream #2 includes information from different sources and viewpoints on how software design is conducted. Although this is a stream of references not immediately active, its use is to help outline the borders between this approach - which is process-based - and other approaches like methodological, user-centred, reuse-based etc. It can be very useful to know the commonalties and differences of other approaches. This stream also provides information for a series of process which cannot be described as purely cognitive. Example references: [61, 41].

Stream #3 includes a big variety of tools that support mostly high-level design activities. Typical references: [3, 21, 76].

2.3 Establishing a context: the software design factors

There is a series of factors that can affect the software design process, like the type of software being produced, the application area, the methods used etc. These factors are termed "software design factors" and are listed and analysed in the next paragraph.

To be able to produce a correct and consistent view of things, the software design factors are seen as a list of variables all of which must have a certain value before the associated studies are considered in this project. For example, the variable application_area could have the value "Management Information Systems (MIS)". It then has to be considered whether a study describing the design of an MIS should be included in this work, and so forth. Another reason to consider software factors as variables is that it is compatible to the way behavioural studies treat such issues in order to measure them, i.e. as independent/dependent variables, and therefore it is easier to include the outcomes of these studies to the framework as long as the variables of interest are instantiated (or justifiably uninstantiated).

At this point in research it is found useful to leave most of these variables uninstantiated and this is justified as follows: the empirical studies that provide the source material of this research focus on a certain context e.g. individual problem solving, analysis of design meetings etc. Leaving some of the software factors uninstantiated gives the opportunity to broaden the scope of the framework (on the contrary, the more factors are instantiated, the more specific the framework becomes). Different design situations can then be included into the framework by combining results from several studies. By careful selection of which variables to instantiate, there are no contradictions in the framework.
The advantages of this merging approach are two:

(a) it is the first study - to the writer's knowledge - that combines findings of disparate studies conducted in different design contexts. Therefore it is tried to create a holistic view of all the different design activities and their support, providing a further insight into software design.

(b) it is the case that in a very complex activity like software design it is not always possible to distinguish the different activity types at a certain point in time because these happen iteratively, opportunistically [35] and unpredictably.

For example, although software design is becoming more and more a social activity (in a 1987 study it was found that a typical computer system developer spent 70% of his time on a large project working with others [31]) atomic design is always part of it. In other words, a designer does both atomic and social/organisational work and therefore all the different design levels are actually subsets of the design process as a whole. Thus, there is no reason why they should be treated differently. On the contrary, all design subprocesses should be considered at the same time, independently of level, as they are all into play during design.

This is also an example of how a careful selection of the variables to be instantiated can instead of leading to conflicting information, enhance the framework. In this case, the parametric nature of the uninstantiated variable/factor design_dynamics enables more empirical evidence to be included and thus enriches the framework.

The main factors that form a design context and therefore affect the designers' activities are listed below.

User participation is a type of design that might or might not take place in practice. However, when it does take place, it makes a difference on how a designer will work. Malhotra et al. [61] report findings from observational studies on design sessions where both a designer and a client (user) were present. The findings are different from cases where there was no user involvement [35, 54, 94, 95]. Therefore it makes a difference for a designer to talk through the user requirements instead of going through requirements or design documents. Thus, the variable user_participation has the value "yes" which means that in the framework negotiation with users is considered as one of the design processes and that is because it is thought of as an essential activity.

Design routine Gero [29] distinguishes between routine and non-routine design. He defines routine design, in computational terms, as "that design activity which occurs when all the necessary knowledge is available". He also subdivides non-routine design into innovative and creative design. Glass [33] makes a similar distinction. It is believed that the case in software design is very much a process of knowledge discovery as reported in a few studies [38].
Therefore the variable design_routine in the context of this study takes the value "non-routine design".

Design history refers to whether a software artefact is designed from scratch and therefore I am referring to the initial design, or the request is to improve an existing design which is usually called a re-design situation. (There can be a few different re-design situations). The variable design_history has the value "initial design". It must be noted here that the distinction between initial design and re-design is not unanimously clear in the literature. In this report the thesis taken is that to have a re-design session some kind of user evaluation/software test must have been done at least once so far.

Application area/problem domain, e.g. "Operational Research"/"The Simson linear programming method". They make a difference in the way a designer approaches a problem. For example, the design of a real-time system follows application-specific rules and probably its own techniques/tools. I will not commit into any specific application area at the moment. Therefore the variables application_area/problem_domain have no value (null).

Design dynamics refers to the number of people who are responsible for the creation of the design. As empirical research shows [18, 95] team design raises issues of multi-agent problem solving and communication which are foreign to atomic design. The organisational impact on communication among or inside design groups is also considerable. Therefore, this is another variable that deliberately stays uninstantiated, in order to include both design situations in the study. It is usually analogously related to the software_scale variable.

Designers' performance plays a large role in designers' actual activities. Curtis et al [18] interviewed personnel from 17 large projects and analysed a series of problems with productivity and quality towards three levels: cognitive, social and organisational. One of these problems is called "the thin spread of application domain knowledge" (for a full description see paragraph "knowledge Interaction") and working on that they found that individual performance is a combination of motivation, aptitude and experience. More specifically, exceptional designers: (a) were extremely familiar with the application domain and they were able to map between the behaviour required of the application domain and the computational structures that implemented this behaviour (b) were skilled at communicating their technical vision to other project members, and (c) usually became consumed with the performance of their projects. These three findings can be summarised by the characterisation of exceptional designers as interdisciplinary since they integrated several knowledge domains that constituted the application domain. This knowledge acquisition often consists of disorganised education acquired on-the-job. This is a point which will be picked up on later in this thesis as it is central in the current investigation.
In another study, Adelson et al. [1] performed a controlled experiment in order to find out how experience in a domain affects a designer's knowledge and skills. As a result, they have identified and analysed designers' behaviours that are affected by their experience with the object being designed and the domain.

The variable `designers_performance` will remain uninstantiated in order to accommodate for designers with various levels of expertise and retain the opportunistic character of the software design process as observed in quite a few studies [36, 56].

**Software type** Following software classifications that are based on the relationship of the software to the environment with which it operates, Gidding's classification [31] is adopted. More specifically, he classifies software in two categories according to the way in which the universe of discourse (the class of the problems to be computed or the problem domain in terms of the rest of this document) and the software interact. **Domain independent** software is distinguished by the independence of the problem statement and the universe of discourse. The essential problem is proving that one has in fact obtained a solution (verification). **Domain dependent** (DD) software is divided into experimental (DDEX) and embedded (DDEM). DDEX software is characterised by an intrinsic uncertainty about the universe of discourse. The essential problem is producing software useful for testing a hypothesis or exploring unknown characteristics of the universe. On the contrary, DDEM software (a model of which is shown in figure (1)) is characterised by interdependence between the universe of discourse and the software.
The use of the software may change both the form and the substance of the universe of discourse and, as a result, the nature of the problem being solved. The fact that most software is domain dependent [31] and the interesting interdependence between the problem domain and its formal description (in computational terms) leads to the choice of this type of software for further study. Moreover, it is considered that the evolution of the problem domain - which characterises DDEM - is closer to the software type of interest to this work. It must be noted here that this choice does not exclude the future possibility of writing experimental software (DDEX) whose purpose is to test a hypothesis concerning the usefulness or effectiveness of a DDEM software assistant. In other words, while the software to be supported is always DDEM, the support tool itself might be a piece of DDEX software (as it is originally generated to test a research hypothesis) which may or may not become DDEM. Bearing this classification, the variable software_type has the value "Domain-dependent embedded".

Software scale concerns the size of the project i.e. programming or rather designing-in-the-large versus designing-in-the-small. It also makes a difference because the sophistication of the employed methods/strategies are analogous to the size of the project. This variable is also uninstantiated.

Technological environment. This is a compound factor comprising of a number of variables, relating to different technological factors which admittedly play their role on the design process. Here is a list of these variables and some typical values of theirs: design_method (e.g. JSD, OOD etc), support_tools (e.g CASE, UIMS), formal_methods (yes/no), programming_environment (Programmer's Workbench 2.0), target_machine (e.g. high-end mainframe) and integrated_system (e.g. CLIPS). All these variables and therefore the compound variable technological_environment stays uninstantated. This results to the inclusion in the framework of any design activity that can be affected by any of these.
technological factors. However, the rationale behind the uninstantiation of these variables is more that at this point of research we are not concerned with technological details in gathering information for the design process, rather than that I have made sure that all the possible combinations of values of these variables will produce a valid context for this research. For example, the process of visualisation of past design experiences is one that could be significantly enhanced by the use of computers, providing that these types of experiences are properly conceived and formalised. His would be an instantiation of the variable support_tools. It could then be valid and worth to survey design activities given this type of support, as the use of computers may cause changes to the design tasks and to the universe of discourse as is the case in domain-dependent embedded software. At the moment, most of the included empirical evidence is derived from design contexts where there was no special consideration of technological impact.

Business status is a factor that implies all the impacts that are external to software but do affect the design process like budget, type of contract, company policies (work practice/characteristics etc..). It is out of the scope of this research to analyse these factors so the business_status variable is uninstantiated.
2.4 Empirical studies of software design

An overview of empirical studies of software design. The contribution of each study will be elaborated later on in the chapter. This section briefly reviews the main such studies ordered by context, initially adopting the categorisation used by Curtis in [19].

2.4.1 By dynamics

2.4.1.1 Individual level

In experimentally created design contexts, Adelson and Soloway [1] studied designers at different levels and types of familiarity in order to find out the effect of domain experience in software design. They found a list of skills and knowledge that designers had developed more or less as part of their experience, such as formation of mental models and simulation. In addition, they found differences in the impact of designers’ experience with the object being designed and the domain in which it is being designed.

Kant and Newell [51] have studied two Ph D. level computer scientists designing an algorithm to the convex hull inclusion problem. The subjects worked mostly in an algorithm problem space, but also, though much less frequently, in a geometry domain space. This sometimes led to the discovery of new and important solution insights. Kant and Newell suggested that the interplay between problem solving and the two spaces permits the process of discovery.

Vitalari [94] has performed an empirical study in order to find out the types of cognitive behaviour and knowledge used by systems analysts to determine information requirements (requirements analysis phase). The main outcome was that most of the analysts’ time was spent setting goals, formulating strategies to achieve the goals, developing hypotheses about the nature of the requirements and searching for clues in the problem statement. The analysts basically formulate and scope out the problem in such a way that the problem space is constrained to reduce the time entailed in searching for a high quality solution.

Visser [93] has presented the results of three observational (field) studies which involved different design tasks, i.e. functional specification, software and composite-structure. The designers’ activities are thoroughly analysed at three levels: planning, adoption of strategies and problem solving. Observed processes include the use of examples and analogy as well as opportunistic organisation.

Rosson et al. [78] performed an exploratory study. They conducted a series of interviews on designers of interactive systems, in a variety of work contexts. The aim was to find out more about the actual practice of well advocated concepts such as user testing, user interface...
consistency, prototyping etc.. They address in detail issues like iteration and user testing, user interface design and idea generation.

One of the most thorough studies of software design is described by Guindon & Curtis in [37]. Three professional software designers were presented with a problem of realistic complexity and the resulting verbal protocols were studied. A plethora of cognitive processes were revealed, ranging from process control strategies and knowledge sources to strategies, heuristics, constraints etc.. Opportunistic behaviour was also apparent in this study as revealed in the constant iteration between levels of abstraction.

2.4.1.2 Social/team level

Empirical research on team design raises issues of multi-agent problem solving and communication which are foreign to individual design. Krasner et al [54] videotaped design meetings of a software project that involved the development of an object-oriented database system. Walz et. al. [95] analysed the resulted protocols. They argue that conflict was the mechanism for facilitating learning. Further on, they recommend consideration of formal techniques for managing conflict to help with knowledge acquisition, sharing and integration. The importance of team problem solving is emphasised by Walz et al. [95] who advocate that multi-agent cognitive processes must be modelled as the fundamental element of team behaviour in design, rather than begin with the more traditional group process elements of role formation, leadership style etc.

Bennet & Karat [4] performed a study on HCI design meetings. Their aim is to gain insight on how to facilitate such meetings. They analysed facilitation in terms of partnership, stages of meetings and team conversations. They also observed how facilitation skills might be developed, respectively.

2.4.1.3 Organisational level

The organisational impact on communication among or inside design groups is also considerable. Curtis et al. [18] interviewed personnel from seventeen large projects and analysed a series of problems with productivity and quality towards three levels: cognitive, social and organisational. The three major problems they identified were: "the thin spread of application knowledge", fluctuating and conflicting requirements, and communication bottlenecks and breakdowns. The "thin spread of application knowledge" refers to the fact that a deep understanding of the application domain and its relationship to system architecture is not widespread in the programming workforce. A main point is the characterisation of exceptional designers as interdisciplinary since they integrated several knowledge domains that constituted the application domain. This knowledge acquisition often consists of disorganised education acquired on-the-job.
2.4.2 Other studies organised by paradigm

In order to cover a wide range of design activities, two more specific types of design are additionally considered. These are participatory design which is a type of design practised in usability-oriented projects and object-oriented design as the dominant paradigm of the 1990s.

2.4.2.1 Participatory design

Malhotra et al. [61] report findings from observational studies on design sessions where both a designer and a client (user) were present. The findings are different but not contradictory to cases where there was no user involvement. They mention three interacting processes: goal elaboration, design generation, and design evaluation. Their studies also showed that design specifications are often incorrect and incomplete with different designers paying more or less attention to different aspects of the design.

2.4.2.2 Object-oriented design

Lubars et al. [60] address the case where in a customer-specific project, the customer provides a large body of documentation of requirements. Describing their experience with the TWCS system, they argue the necessity of a model of the requirements in such a case (the size of the system is obviously large). In this context, they advocate the disambiguation and completeness-check of the bulk of the requirements into fragments that correspond to subproblems (bottom-up approach), and then the use of scenarios in order to specify the interrelationships between these fragments (top-down approach). It seems then that in any case - i.e. whether a methodical requirements specification is done, or the size of the problem does not necessitate it - there is a big deal of problem solving effort early at this phase to define the problem, reformulate it, and constraint its problem space.

More recently, Fischer et al. [28] proposed a cognitive model of object-oriented development based on analysis of relevant experiences and empirical studies. The model emphasises a longer period of time than its predecessors and focuses on the interaction among many individuals and different kinds of involved artefacts. Its main aspects are evolution, reuse and redesign, and domain orientation.

Earlier work on the organisation of software design activities showed that notions of a predefined sequence were misguided and that, not only is design characterised as a loosely structured process, but that designers are able to handle different levels of the abstraction at the same time. Pennington et al. [71] examined object-oriented design in relation to previous work and confirmed earlier findings. A particular point made was that there were two critical factors, those of designer experience and movement between levels of abstraction. The
experienced designer is more inclined to change level frequently although other factors also apply.

2.4.3 Summary of findings

The practice of software design is very diverse, and some of the factors that contribute to that diversion are the size of the project (and subsequently how it is managed, manned etc.) the plethora, diversity and availability of support tools and to mention a few. The previous section provides a general overview of software design activities. Therefore, not all the empirical findings mentioned in the previous section apply to every software design situation.

The full set of cognitive processes that were reported in the above studies is used to produce the Software Design Support Framework (SDSF) which is a summarising framework of software design practice, and is described in section (2.6).

As a summary, the main points that come out of this survey are as follows:
* software design is an ill-defined problem solving activity
* designers exhibit opportunistic behaviour
* constant interaction among multiple levels of abstraction
* combination of multiple knowledge types
* incomplete and fluctuating requirements

It is also apparent that the level of expertise is significant factor in designers' behaviour.

2.4.4 Classifying processes

In this section, software design processes encountered in the literature are analysed and classified for the purpose of providing a summarising framework.

Design is solely a problem solving activity. More specifically, the class of problems it deals with are ill-structured. These are distinguished from well-structured problems [83] by four characteristics (see section (5.2.2) for a more detailed discussion). Furthermore, a social study of design in practice reveals that its identity against other ill-structured problems like writing, lies upon the impact of external or ecological factors like intra- or inter-team consensus, communication, organisational behaviour etc.

In this thesis it is tried to identify the constituent parts of this problem solving activity and analyse them as clearly as possible. At the same time, the analysis is done under a set of perspectives i.e. cognitive, social, organisational, engineering, although most of the material belongs to cognitive studies of design like [36]. The categorisation is done mainly by extracting and emphasising the major/most important features of every perspective in order to see them better and investigate their potential support. In this paragraph, the processes that are included in SDSF are described in more detail.
Individual problem solving is regarded as the core activity of high-level software design, and by many authors it is presented as any other solving task of problem contexts like Mathematics etc. It is the activity where a designer studies the problem and tries to come up with a solution to it. It is that part of software design that will actually produce the initial architecture of the required system usually after a series of experimental attempts. This abstract description of the software system will be subject to further analysis or decomposition or any type of progression (elaboration) to lower levels of abstraction (at least one) which will lead to program (data) specifications that are codable.

Empirical studies reveal the problem-solving nature of software design. According to the definitions given by Malhotra et al. [61], a problem state exists when a human, or other goal-oriented system, has a goal but no immediate procedure that will guarantee attainment of the goal. The goal may be a satisfaction to be achieved or a dissatisfaction to be alleviated. Problem solving occurs in moving from a problem state to a non-problem state. In problem solving, then, a person begins in an initial state, uses transformations that move him/her from one state to another, and ends in a final state.

Further on, Gundon [40] justifies the ill-structure of design as a problem (see also table (5a) in chapter 5) by making use of some of its characteristics, including:

- Incomplete and ambiguous specification of the problem
- No stopping rule - no definite criteria to evaluate whether a solution is reached
- Many sources of knowledge (problem spaces) that cannot be determined in advance and need to be integrated
- No exhaustive, innumerable list of operators to reach a solution and absence of predetermined solution path from initial state to goal state

In Curtis's three-level (individual, team, organisational) study of software design [19], problem-solving was found eminent not only at the individual but also at the team level where Curtis even notes the need for a new model (of it).

Elaboration and understanding of the requirements refers to any activity that aims to decrease the incompleteness and ambiguity of the initial problem statement [40]. In particular, Gundon mentions:

(a) the inference of constraints which are not explicitly given in the requirements but can be deduced as a logically necessary or possible inference (abduction) from the inferential
specification and from one's knowledge of the problem domain. Inferred constraints include inferred objects, inferred relationships between objects, inferred properties of an object, inferred actions of an object, and inferred test cases;
(b) the addition of new requirements which are desirable but optional. The addition of such requirements may reduce the space of design possibilities and also play a critical role as preferred evaluation criteria or stopping rule.

Understanding the requirements may include testing the consistency of the requirements with the designers' own general knowledge. Designers also identify and abstract the critical points from the requirements and give them priority.

Vitalari [94] summarises the processes that act upon requirements as
1. Anticipation
2. Elicitation
3. Assurance
4. Specification
The first three of these processes require extensive interaction with the user and are going to be analysed in the "User Involvement" section. However, they may also exist in a situation where the designer is left alone with a (probably informal) statement of the problem to do all the work by him/herself. Requirements anticipation is defined as the process where the designer hypothesises that particular requirements are relevant based on their previous experience with other similar systems and knowledge about the field. This corresponds to the process of the addition of new requirements described above. Similarly, requirements elicitation and assurance correspond to the processes implied by understanding of the requirements and are also described previously. Requirements specification is addresses in the following paragraph.

Lubars et al [60] address the case where in a customer-specific project, the customer provides a large body of documentation of requirements. Describing their experience with the TWCS system argue the necessity of a model of the requirements in such a case (the size of the system is obviously large). In this context, they advocate the disambiguation and completeness-check of the bulk of the requirements into fragments that correspond to subproblems, and then the use of scenarios in order to specify the interrelationships between these fragments. It seems then that in any case - i.e., whether a methodical requirements specification is done, or the size of the problem does not necessitate it - there is a big deal of problem solving effort early at this phase to define the problem, reformulate it, and constraint its problem space. A natural question at this point is whether software documentation produced at requirements specification but also software documentation in general is a side issue in software development and to what extent it actually helps in the real design activities.
It must be said that designers' activities are found to be "...highly iterative, interleaved, and loosely ordered..." [38]. The evidence so far is that analysis and design are highly interleaved and there is no point in separating them.

Handling conflicting and fluctuating requirements: Curtis et al. [18] studied among others the problem of fluctuating and conflicting requirements by interviewing personnel from 17 large projects. The study was very thorough and examined the issue through a number of perspectives. The results are summarised as follows:

- **At the business milieu**: Product requirements fluctuated more frequently when (a) different customers had separate needs, or (b) the needs of a single customer changed over time, following the evolution caused by the introduction of a customised system. This evolution happened in terms of the growth of knowledge from the users' side of (i) the application domain and (ii) the system's capability.

- **At the company level**: On projects producing commercial products, internal company groups such as the marketing department, often acted as a customer. They could add conflict into requirements definition since their requirements occasionally differed from those of potential customers.

- **At the project level**: Unstable requirements, when caused at the project level, usually resulted from the absence of a defined mission. Without a sense of mission the motivation for the project could not be translated into clear product requirements.

- **At the individual level**: New requirements frequently emerged during development since they could not be identified until portions of the system had been designed or implemented. Many designers thought that requirements should act as a point of departure for clarifying poorly understood functions interactively with the customer. They argued that specification should not be formalised any faster than the rate of uncertainty about technical decisions is reduced. A customer representative's statement is characteristic: "You will never really be able to specify enough detail. It doesn't matter how. You can even take the actual system and write the specs around it and still come out wrong .. The specifications are something you've got to take on trust."

In another study, Walz et al. [95] videotaped design meetings of a software project that involved the development of an object-oriented database system, and analysed the resulted protocols. They argue that conflict was the mechanism for facilitating learning. It was not a debilitating factor needing to be suppressed in the software design team. Further on, they recommend consideration of formal techniques for managing conflict to help with knowledge acquisition, sharing and integration. They suggest that at least one person within the group serve in the capacity of a facilitator of programmed conflict. This individual would receive formal training in the DADP(1) or the SAST(2) as well as training in dialectic thinking and philosophy.
It must be noted that in the participatory design sessions studied by Bowers et al [6] there was no real conflict observed. Explicit disagreements on the part of the user to suggestions made by designers or vice versa were rare. Designers and users are attentive to each other's skills and abilities and so do not directly confront each other by denying the worth of each other's contributions. Refusals, disagreements and resisting suggestions for redesign take on a much more indirect form. Therefore users and designers are orienting to each other's skills and knowledge in ways which obviate the need for direct requests or refusals.

(1) The Devil's Advocate Decision Program is one of the techniques for programming conflict into organisational decision making processes where an individual or group plays the role of critic in order to help a decision maker test the assumptions and the logic of the ultimate decision.

(2) The Strategic Assumption Surfacing Technique offers a method by which facilitated groups can identify and resolve underlying differences and similarities.

Handling uncertainty and breakdowns: Uncertainty in the context of software design might be perceived as incomplete, ambiguous, possibly incorrect or totally unknown information about a certain aspect of the problem domain. In a generalised way, it can be thought of as lack of knowledge of "the universe of discourse" [31] i.e. of a particular domain. Uncertainty is one of the causes of the so-called breakdowns in the software design process. Hence, in this paragraph I am not looking at the communication or co-ordination breakdowns that occur in team processes.

Breakdowns are more or less difficulties found by software designers in developing a control strategy. Guindon et al. [35] observed a series of breakdowns by studying the design of a problem of realistic complexity. She found three classes of breakdowns; one is caused by lack of knowledge, another is caused by cognitive limitations and a third one is caused by a combination of these two factors.

Before listing the main breakdowns, let's clarify the notion of a design schema. A design schema is basically a representation of a past experience. According to Guindon et al. [35] a design schema is composed of a description of the conditions under which a solution plan is relevant. These conditions contain a representation of critical features in the problem domain. The problem statement, while being comprehended, is mapped on this representation of the critical features. The design schema whose conditions are best fit by the understood problem statement is retrieved. A design schema is believed to express the generalisations or abstractions made from specific experiences of software design. Once recognised as relevant, a design schema can be applied during the design process. The design schema
contains a solution plan to guide the decomposition of the problem into subproblems, each subproblem with its own design schema. As a consequence, the design schema influences the control of the design activities through a recognition process and through the execution of a solution plan. The design schema also supports the storage and retrieval of intermediate solutions, and as a consequence, reduces working memory load during design and increases the probability that partial solutions and postponed subproblems will be retrieved when needed. This is because the design schema embodies large chunks of structured knowledge [37].

The main breakdowns observed are: (1) lack of specialised design schemata; (2) lack of a meta-schema about the design process leading to poor allocation of resources to the various design activities; (3) poor prioritisation of issues leading to poor selection of alternative solutions; (4) difficulty in considering all the stated or inferred constraints in defining a solution; (5) difficulty in performing mental simulations with many steps or test cases; (6) difficulty in keeping track and returning to subproblems whose solution has been postponed, and (7) difficulty in expanding or merging solutions from individual subproblems to form a complete solution.

Simulations in the problem domain: According to Guindon et al. [36] one of the designers' activities was the retrieval or the simulation of scenarios in the problem domain. Five main uses for the retrieval and the simulation of scenarios were observed in the protocols:
(1) Understand the given requirements with scenarios.
(2) Understand the inferred requirements with scenarios.
(3) Solution development with scenarios
(4) Unplanned discovery of new requirements.
(5) Unplanned discovery of partial solutions.
All the designers in this study supported the simulations of problem domain scenarios by using external representations. Note making is addressed in a later paragraph. Another important finding is that simulations of scenarios in the problem domain can lead to the discovery of critical knowledge not only about the solution but also about the problem goals and evaluation criteria.

Simulations of the design solution. These help uncover various types of bugs [36]: (1) inconsistencies with other parts of the solutions; (2) incompleteness of the solution with respect to the rest of the design solution; (3) inconsistencies with the given or inferred requirements; and (4) incompleteness with respect to the rest of the design solution. All these simulations relied on external representations because, first, the external representation is the artefact, and second, they are cognitively taxing. As mentioned previously, solution simulations were done in terms of test cases based on problem domain knowledge. There
were two types of simulations: executions with specific test cases and symbolic executions using variables instead of specific cases.

An interesting observation is that the mental simulations of the solutions were shallow, that is, most were restricted to one level of abstraction and one subsystem. They did not involve the simulations of subsystems embedded within the simulation of a higher-level subsystem. This is not surprising given the severe limitations that working memory poses on the processing of embedded structures. The need to remember the values of the variables and the return locations in the higher-level system (i.e. the context) when simulating an embedded subsystem probably precludes simulations of solutions at more than one level of abstraction. Of course, this limitation was partly alleviated by the use of external representations, but not completely. Likewise, the solution simulations often triggered the inferences of new requirements and the discovery of partial solutions at arbitrary points and levels of abstraction in the solution. These greatly contributed to the observed opportunistic character of the design process [36].

Adelson and Soloway [1] performed a controlled experiment, studying designers of varying expertise in contexts of varying difficulty and familiarity. They also observed simulations of mental models. Mental models could be described as pre-mature descriptions of external solution representations. Therefore, they are internal representations close to the line between problem and solution. They are further described later in this chapter.

Formulation of solution models: This activity is very important because it is the one where the designer actually lays down a solution. In a way, all the other activities in the framework mean to actually assist this process which results to the production of a design, or a design model. Let's analyse all the subsequent processes that are implied here, in turn:

Mental and other models. Adelson and Soloway [1] describe the mental models as internal working models of the design-in-progress. They are defining a model as a representation which is capable of supporting mental simulations of the design in progress. It is a representation in which the commands of the system being designed can be viewed as actions. Further evidence of the existence of these internal models is that frequently in the empirical studies they served to generate external models in the form of sketches of a design-in-progress.

Mental models are not the only type of models that can be found in design contexts. Design notation instances are typical models of the solutions and they are addresses separately. User and task modelling is another crucial early activity in the design of interactive systems for which there's no empirical evidence on how exactly is conducted in practice and how it is related to high-level software design.
Annotation: It can be found as
(a) external representations of the solution to:
    -> express the design solution itself
    -> perform mental simulations of the solution
    -> uncover missing information and to ensure completeness of the solution
In Adelson and Soloway's study [1] these sketches typically consisted of some initial node with arcs from it to other nodes in the sketch. The initial node depicted an action or command of the system and the arcs specified the actions that could follow as a result of the initial action.
(b) design notations to:
    -> transform the informal specification of the requirements into a design solution
    -> develop the design solution.

Problem solving strategies: The use of design schemata has already been mentioned as a provision of help and expertise. Another type of schemata, domain-independent, problem solving schemata are also mentioned in the literature. These include the divide-and-conquer and generate-and-test, although their use must be said that predominates design problems of limited complexity. When designing an object the goal state is specified as a behaviour, but not the current state. That is specified as a set of partially designed mechanisms which will eventually exhibit the behaviour. The simulation of a mental model allows the current state to be specified as behaviour. This enables the designer to solve the problem by assessing the nature of the differences between the current and the goal states and then choosing a means to reduce the difference. This is referred to as a solution through means-ends analysis.

It was also observed by Adelson and Soloway [1] that the nature of each designer's mental model changed as his design progressed. Each designer's model started out at an abstract level of representation and progressed to a concrete one. Additionally, only one level of representation is focused on at a time and each level of representation is only a bit more detailed than the next. This is termed systematic expansion.

Evaluation of solutions: Regarding the evaluations of alternative solutions, it was found [36] that designers rarely retrieve more than one alternative solution for a problem. But if the designers retrieved more than one solution to a subproblem they very rapidly selected one of them. The rejections of alternative solutions were made very rapidly, without developing them in depth.

Use of design methods and notations: A design method, by definition, dictates or suggests a sequence of activities to be performed, and therefore is a prime influence on the planning and control of the design process. A design method provides a sequence of operators and associated tests to apply these operators, for the transformation from an informal specification
of the requirements to a design solution. Therefore, design methods provide operator sequence knowledge and control knowledge for the application of these operators. This knowledge is specific to software design tasks, but independent of the problem domain (though different design methods are best suited to different classes of problems).

Knowledge interaction/learning: It is according to Guindon an integral part of the problem solving nature of design. However, the terminology used here serves mainly the detailed analysis of processes and that's why it is treated separately. Knowledge interaction is mainly a by-product parallel process embedded in the evolution of the design.

Combination of different types of knowledge implies the process where an individual designer combines different types of knowledge in order to create his own part of the design i.e. the architecture of the software system. For example, a designer that designs a spreadsheet application has to somehow learn about general accounting, how accountants practice logistics, and how they would interact with a spreadsheet system i.e. problem domain, task and perception knowledge. For a thorough investigation of the types of knowledge that contribute to a design, see [22]. For a set of knowledge types involved in software design, see [40].

Knowledge increase and evolution as design progresses refers to the process where the knowledge of the individual designer about different aspects of the design increases as the software project goes on. According to Guindon [40], software design is an ill-structured problem, and at the point where a designers reaches a solution (which is a subjective decision), learning stops. Therefore there is an assumption that design is a continuous learning process. The hypothesis at this point is that these two processes are interleaved.

Capture of unstructured information/knowledge refers to the process where designers accidentally discover new knowledge about the system or ideas from external sources [18] e.g. reading an article in a newspaper, discussing a subject in the corridor, being handed a requirements-related document etc.. The incorporation of this type of unstructured information into the design might be useful to be treated as a different process.

Spread of application knowledge across the design agents. The "thin spread of application knowledge" is a problem notified by Curtis et al. [18] and refers to the fact that a deep understanding of the application domain and its relationship to system architecture is not widespread in the programming workforce. Users are treated as design agents, as well. This is another issue which is looked at in more detail in the following chapters.

Verification and validation of constraints. In a design situation there are hard rules and soft rules. To actually produce a creative piece of work you probably have to break some of
these rules [22]. However, a designer needs to verify and try out some rules/constraints and check his/her design against them in various levels of strictness.

**Use of heuristics.** Very often organisational strategies are posed to the software project and have to be followed, as a piece of software is more and more viewed as a product rather than technological achievement. Heuristics are meant to as practical recipes that have worked before and will get the designers to the solution more quickly.

**Design agenda.** As design might take long, a need for an agenda is essential for the life of the project [2]. Furthermore, as design is a cognitively intensive activity, it seems that assistance of human memory could help designers a lot. There is some recent research on this issue and it will be looked at in more detail.

**Retrieving labels for plans.** There are situations (see [1]) in which the designer has previously worked out a solution to an aspect of the problem but does not concern himself either with working out or recalling the implementation details at that point. He then creates labels for elements of the problem which have a solution stored as a plan in memory. The plans themselves will be retrieved later when the level of detail of the design matches the level of detail of the plan. Before that time the label is retrieved as a place holder of that plan.

**Exploitation of past experiences.** Expertise occupies a large proportion of creative design. Past *experiences* can be thought of in terms of multiple views of a design: e.g. domain representation, "design schemata" (i.e. internal representations of past experiences), documentation of software or user interface designs etc. Re-use can happen on a variety of dimensions e.g. previous designs of the same designers in different contexts, previous designs of different people in the particular domain of interest, designs in different contexts that have similarities in e.g. problem solving or user interface strategies/techniques, or even a re-design of an existing system.

**Recording of design rationale.** The recording of design rationale is quite essential as it (a) helps designers clarify their thoughts about the systems they're building and (b) serves to document design decisions to be re-used at some later time [2]. Design rationale usually takes the form of questions, options, and criteria of justification [61].

**Collaboration with other designers.** Design is a collaborative activity. De Marco and Lister found that a typical computer system developer spent 70% of his time on a large project working with others (cited in [31]). Therefore communication with other designers/project members is a key subprocess.
User Involvement: Direct user involvement can be done in different ways in the design phases. More importantly, the two types of processes observed by Guindon [37] and Malhotra et al. [62] are corresponding. Therefore they could have been categorised under the same activity. The only reason that user involvement is presented as a different process is (a) to emphasise its importance as a design activity and (b) to show its importance as a communication process treating the user as an active design agent.

2.4.5 A design process framework

The analysis and classification of processes presented in the previous section is used in this section to form a model of activities that covers most aspects of design work. That model framework is presented below in three parts.

<table>
<thead>
<tr>
<th>Design activities</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General activities</strong></td>
</tr>
<tr>
<td><strong>Subprocesses</strong></td>
</tr>
<tr>
<td>Elaboration and understanding of the requirements</td>
</tr>
<tr>
<td>Handle conflicting requirements</td>
</tr>
<tr>
<td>Handle uncertainty or breakdowns</td>
</tr>
<tr>
<td>Simulations in the problem domain (internal and external)</td>
</tr>
<tr>
<td>Simulations of the design solution (internal and external)</td>
</tr>
<tr>
<td>Formulation of solution models</td>
</tr>
<tr>
<td>Methodology independence</td>
</tr>
</tbody>
</table>

Figure 2b-a: Design process framework – part 1
### Design activities

<table>
<thead>
<tr>
<th>General activities</th>
<th>Subprocesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge interaction/learning</td>
<td>Combination of different types of knowledge</td>
</tr>
<tr>
<td></td>
<td>Knowledge increase and evolution as design progresses</td>
</tr>
<tr>
<td></td>
<td>Capture of unstructured information/knowledge</td>
</tr>
<tr>
<td></td>
<td>Spread of application knowledge across the design agents</td>
</tr>
<tr>
<td>Intervention of rules</td>
<td>Verification and validation of constraints</td>
</tr>
<tr>
<td></td>
<td>Use of heuristics or strategies</td>
</tr>
<tr>
<td>Process management</td>
<td>Design agenda and notes</td>
</tr>
<tr>
<td></td>
<td>Information processing</td>
</tr>
<tr>
<td></td>
<td>Retrieving labels for plans</td>
</tr>
</tbody>
</table>

**Figure 2b-b: Design process framework – part 2**
### Design activities

<table>
<thead>
<tr>
<th>General activities</th>
<th>Subprocesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualisation of the design</td>
<td>Exploitation of past experiences</td>
</tr>
<tr>
<td></td>
<td>Recording of design rationale</td>
</tr>
<tr>
<td>Communication</td>
<td>Collaboration with other designers</td>
</tr>
<tr>
<td></td>
<td>Handle communication and collaboration breaks</td>
</tr>
<tr>
<td></td>
<td>Collaboration with the business milieu</td>
</tr>
<tr>
<td>User involvement</td>
<td>Goal statement</td>
</tr>
<tr>
<td></td>
<td>Goal elaboration</td>
</tr>
<tr>
<td></td>
<td>(Sub)solution explication</td>
</tr>
<tr>
<td></td>
<td>Agreement on solution</td>
</tr>
</tbody>
</table>

![Figure 2b-c: Design process framework – part 3](image-url)
2.5 Software tools for software design

2.5.1 Choice of tools

The choice of the appropriate tools is not an easy task. There exist numerous tool classifications each representing another perspective, e.g. see [74]. Other problems that relate to the choice of tools include the often inadequate information about them, and their continuous evolution. In the end, (a) diverse tools are included, (b) those tools are chosen arbitrarily, (c) it is decided not to make a critical appraisal of them as most vendors do not claim design cognitive support anyway. Let’s not forget that most CASE tools simply implement a methodology, i.e. to an extent they cannot alleviate the inherent to the methodology cognitive barriers but can indeed offer extra cognitively supportive capabilities, e.g. multiple design viewpoints, usable re-editing facilities etc.

The selected tools must be as diverse as possible so that they cover a wide spectrum of philosophies without losing the scope of assisting software design. It is tried to identify a series of research areas or domains and include at least one tool out of each of them. Therefore here is a list of domains and characteristic tools that are chosen:

<table>
<thead>
<tr>
<th>Atomic level</th>
<th>Tool</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) OSC</td>
<td></td>
<td>Domain Analysis</td>
</tr>
<tr>
<td>(2) CATALOGEXPLORER, etc.</td>
<td></td>
<td>AI</td>
</tr>
<tr>
<td>(3) LaSSIE</td>
<td></td>
<td>Computer Science - AI</td>
</tr>
<tr>
<td>(4) Eiffel</td>
<td></td>
<td>Object-orientation</td>
</tr>
<tr>
<td>(5) A UIMS (x)</td>
<td></td>
<td>HCI</td>
</tr>
<tr>
<td>(6) A CASE tool (x)</td>
<td></td>
<td>Software Engineering</td>
</tr>
<tr>
<td>(7) x</td>
<td></td>
<td>Re-use of design components</td>
</tr>
<tr>
<td>(8) x</td>
<td></td>
<td>Human Information Processing acceleration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Social and organisational level</th>
<th>Tool</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>(9) rBIS</td>
<td></td>
<td>Multi-user Hypertext and Real-time Groupware</td>
</tr>
<tr>
<td>(10) COQ</td>
<td></td>
<td>Decision Support</td>
</tr>
<tr>
<td>(11) x</td>
<td></td>
<td>Requirements Engineering</td>
</tr>
<tr>
<td>(12) x</td>
<td></td>
<td>Co-operative UIMSs</td>
</tr>
</tbody>
</table>

Table 2a: List of software tools and their associated domains
It should be therefore be expected that the framework will include more and more tools. As a consequence, more types of support will be included and perhaps new structuring of the "design" part of the framework. At this point of time, three tools were chosen out of the above list to be included in the framework. The choice was arbitrary.

2.5.2 Description of tools

Tool #1

NAME: OSC (Outil de Suivi de Conception) [2].

CONTEXT: An extensible tool shell that lets designers capture and manipulate decisions independently of method.

EMPHASIS: Argumentation.

MODEL: Apart from being method-independent, it is also model-independent in the sense that it does not explicitly adopt any specific design model. However, it does support certain types of iteration e.g. design decisions leading to new problems to be solved, and the like. It implies quite a flexible model.

GENERAL APPROACH: It uses current technology on databases and OOP to set up a database of problems, decisions and assertions) plus an agenda as well as a rule base that ensures consistency of design and offers manipulation capabilities of every kind to the designer. These capabilities enable designers to tailor the environment to their own needs.

MAIN CAPABILITIES:

* Extensibility of the database schema

* Captures a wide view of design components.

FURTHER COMMENTS: Its extensible nature brings to bear Fischer's work which is in the same vein and they both advocate the use of a generalised design-support tool that gives the designer(s) the ability to extend it thereby specialising and tailor the tool towards their specific application. This tool is more generic, though. However, the retrieval of database information could perhaps be more advanced.
NAME: LaSSIE (Large Software System Information Environment) [21].

CONTEXT: LaSSIE is a tool that provides for the description of a large software system by multiple views. It thereby covers a wide spectrum of phases in the software life-cycle, from modelling the problem domain to managing the architecture of the system and to programming and coding.

EMPHASIS: The emphasis is placed on fighting the inherent invisibility of large software systems, i.e. the fact that the structure of software (unlike that of buildings or automobiles) is hidden.

MODEL: Visibility is achieved through the ability to re-use software components by providing sophisticated retrieval and update mechanisms. The adopted model is based on objects and actions and facilitates the visibility of such components at any stage of development. Examples of such components are: entries of the domain, parts of the system architecture, "customer" features, or code structures.

GENERAL APPROACH: It includes a large knowledge base, a semantic retrieval algorithm based on formal inference, and a user interface that incorporates a graphical browser and a natural language parser. To represent knowledge they use the KANOR knowledge representation language which is frame-based with incorporated inheritance capabilities and added-on classification of design elements. It seems quite appropriate for the specific telecommunications system it is applied to.

MAIN CAPABILITIES:

* Makes software "visible" and therefore less complex.

* Provides very sophisticated semantic retrieval of software system components.

* Two different user interfaces (DMI and NLI)

FURTHER COMMENTS: Remarkable combination of Software Engineering and AI. A big contribution to (a) the "re-using past experiences" part of creative design (b) maintaining the domain ontology and concepts throughout the project stages. Would it work for more general software systems?
**Tool #3**

**NAME:** riBIS [76].

**CONTEXT:** Software design as a social activity: a real time group hypertext system which allows a distributed set of users to simultaneously browse and edit multiple views of a hypertext network. The aim is to assist design argumentation.

**EMPHASIS:** Fulfilling the requirements of a real-life design session: brainstorming interleaved with individual work in terms of recording design rationale.

**MODEL:** A two process model including (a) brainstorming and (b) issue resolution which follows the IBIS (Issue Based Information Systems) model and process which in turn includes issues, positions and arguments.

**GENERAL APPROACH:** An riBIS session is a set of users working together. At any time, a user can choose one of two interaction modes with the other people in the session: loosely coupled (LC) mode or tightly coupled (TC) mode. LC mode supports parallel activity; TC mode is for highly focused activity and provides an environment in which the group can see the same subgraph, discuss a selected node, and edit an item in real time.

**MAIN CAPABILITIES:**

* Very efficient analysis of graphs (focusing on a subgraph etc.).

* Accommodation of remote design sessions.

* Turn taking protocol for the Tightly coupled (intensive) appropriate for getting group consensus and feedback, while one person is in control of the mouse.

**FURTHER COMMENTS:** Early usage of the tool produced mixed emotions by users. It seems that there is a slight problem of learnability. However, it is probably the first serious attempt to combine the two worlds of (a) hypertext that is aimed at supporting groups of people and (b) real-time groupware.
2.6 The Software Design Support Framework (SDSF): how well tools support design activities

2.6.1 Aims of the SDSF methodology

The aim of the SDSF (Software Design Support Framework) and the associated methodology is to:
(a) structure the literature research process efficiently
(b) enable us to acquire a holistic view of the state of support of current tools
(c) provide the basis for the design of experiments by outlining trends and weaknesses on similar studies or tool capabilities

In particular, papers like [19] outline the need to focus empirical research on work practice. This piece of work can be seen as a proposal to how to go about doing that.

2.6.2 Outline

Two of the points of emphasis in this research i.e. (a) better understanding of the software design process and its description and (b) close examination of the current tool support for it are chosen to be included directly in the construction of the framework.

The first point, which is rather theoretical, includes the continuous attempt to understand, describe and possibly model the design process and has led to the production of a structure of design activities. If one considers the design of a piece of software as a single activity or process, then this can be divided into lower level sub-processes and this can happen again and again until a set of elementary sub-processes is reached. In its original version (i.e. in the form that is presented in this and the next section) the framework includes two levels of processes: general activities and sub-processes. Later on (as shown in section 2.9) sub-processes were further analysed and a third level was added, that of elementary processes. The process structure is not advocated to provide an exhaustive record of such processes. However, it does try to include most of the major empirical evidence on software design.

The second point is more practical and deals solely with tools, support environments etc. A set of such tools have been analysed to see how well they support that process structure. Although these two streams of work are explicitly distinct and progressed separately, they are on purpose kept discretely close, as there is a strong interdependence between them. More specifically, the cross-reference of the process structure and the tool capabilities can yield most useful information and that is the raison d'etre of the software design support framework. On the one hand, the type and quality of support should specifically refer to certain design
sub-processes and on the other hand, sub-processes that are inadequately supported should be identified right away.

The software design support framework is presented in two forms: an analytical and a synoptic. It has two major components which on the figures are distinguished by a thick line: a theoretical which is called "Design activities" and a practical, named "Tool support". "Design activities" consists of a tabular form of the process structure expanded from left-to-right and from top-to-bottom. "Tool support" includes a list of tools confronting the design activities and indicating the level of support with an appropriate colouring. The analytical form of the framework also includes for every design sub-process a series of practical ways to implement a support for that sub-process on the computer. Therefore it provides further information of how different tools approach design problems often reaching technical level of detail.

2.6.3 Analytical form

Due to space problems, the framework is presented in three portions, as Design support status (1), (2) and (3).

<table>
<thead>
<tr>
<th>Symbolism</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>Full support</td>
</tr>
<tr>
<td>~</td>
<td>Limited support</td>
</tr>
<tr>
<td></td>
<td>No support</td>
</tr>
<tr>
<td>Small positive natural number</td>
<td>Annotation</td>
</tr>
<tr>
<td>U</td>
<td>Unknown at present time</td>
</tr>
<tr>
<td>NA</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Table 2b: Explanation of the symbolisms used on SDSF
<table>
<thead>
<tr>
<th>Design activities</th>
<th>Tool support</th>
<th>Possible type of support</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>OSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LaSSIE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rBIS</td>
</tr>
<tr>
<td>General activities</td>
<td>Subprocesses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elaboration</td>
<td>Inference of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and understanding</td>
<td>constraints</td>
<td></td>
<td></td>
</tr>
<tr>
<td>of the requirements</td>
<td>Addition of</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>new requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Browsing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Querying</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Manipulation facilities: editing, reuse</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Relation to problem solutions or other design components</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Single representation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple representations/ mappings from one to the other</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Abstraction of key points</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual problem solving</td>
<td></td>
<td>Resolve via significance rating</td>
<td></td>
</tr>
<tr>
<td>Handle conflicting and fluctuating requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handle uncertainty and breakdowns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulations in the problem domain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulations of the design solution</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formulation of solution models</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Methodology independence</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2c-a: Analytical form of the Software Design Support Framework (SDFS) – part 1
<table>
<thead>
<tr>
<th><strong>Design activities</strong></th>
<th><strong>Subprocesses</strong></th>
<th><strong>Possible type of support</strong></th>
<th><strong>Tools</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General activities</strong></td>
<td>Combination of different types of knowledge</td>
<td>Organisational</td>
<td>OSC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Domain specific</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Perception</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>General design</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Architectural</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Feature (program)</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>Knowledge interaction/learning</td>
<td>Knowledge increase and evolution as design progresses</td>
<td>Types of representation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard logical language</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Production system</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Frame-based system</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Semantic retrieval</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Inference</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Classification</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Capture of unstructured information/knowledge</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Spread of application knowledge across the design agents</td>
<td></td>
</tr>
<tr>
<td>Intervention of rules</td>
<td>Verification and validation of constraints</td>
<td>Design rationale consistency check</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use of heuristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process management</td>
<td>Design agenda and note taking</td>
<td>Decisions to be justified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Information processing</td>
<td>Assertions to be verified</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retrieval of labels from plans</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) Implicitly, i.e. in terms of representation of design rationale.

Figure 2c-b: Analytical form of the Software Design Support Framework (SDSF) – part 2
<table>
<thead>
<tr>
<th>Design activities</th>
<th>Tool support</th>
<th>Possible type of support</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>General activities</td>
<td>Subprocesses</td>
<td></td>
<td>OSC</td>
</tr>
<tr>
<td>Use of past experiences</td>
<td>Types of experiences</td>
<td></td>
<td>LaSSIE</td>
</tr>
<tr>
<td></td>
<td>Design rationale</td>
<td></td>
<td>rBIS</td>
</tr>
<tr>
<td></td>
<td>Requirements components</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Design segments</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visualisation of the design</td>
<td>Data model</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Object-oriented</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hypertext</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hypermedia</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multimedia</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recording of design rationale</td>
<td>Types of exploitation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Re-use</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adaptation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>Collaboration with other designers</td>
<td>Tightly coupled (TC)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Loosely coupled (LC)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Learning (both the design and the tool) by watching an expert in TC mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Handling of argumentation e.g. forming consensus etc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Face-to-face</td>
<td></td>
</tr>
<tr>
<td>User involvement</td>
<td>Goal statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Goal elaboration</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Sub)solution explication</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Agreement on solution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2c-c Analytical form of the Software Design Support Framework (SDSF) – part 3
### 2.6.4 Synoptic form

<table>
<thead>
<tr>
<th>Design activities</th>
<th>Tool support</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General activities</strong></td>
<td>Elaboration and understanding of the requirements</td>
<td>OSC</td>
</tr>
<tr>
<td></td>
<td>Handle conflicting requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Handle uncertainty or breakdowns</td>
<td></td>
</tr>
<tr>
<td>Individual problem solving</td>
<td>Simulations in the problem domain (internal and external)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Simulations of the design solution (internal and external)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Formulation of solution models</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Methodology independence</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2d-a: Synoptic form of the Software Design Support Framework (SDSF) – part 1
<table>
<thead>
<tr>
<th>Design activities</th>
<th>Tool support</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>General activities</td>
<td>Subprocesses</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combination of different types of knowledge</td>
<td>OSC</td>
</tr>
<tr>
<td></td>
<td>Knowledge increase and evolution as design progresses</td>
<td>LaSSIE</td>
</tr>
<tr>
<td></td>
<td>Capture of unstructured information/knowledge</td>
<td>rIBIS</td>
</tr>
<tr>
<td></td>
<td>Spread of application knowledge across the design agents</td>
<td></td>
</tr>
<tr>
<td>Knowledge interaction/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intervention of rules</td>
<td>Verification and validation of constraints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use of heuristics or strategies</td>
<td></td>
</tr>
<tr>
<td>Process management</td>
<td>Design agenda and notes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Information processing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retrieving labels for plans</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2d-b: Synoptic form of the Software Design Support Framework (SDSF) – part 2
### Design activities

<table>
<thead>
<tr>
<th>Subprocesses</th>
<th>OSC</th>
<th>LaSSIE</th>
<th>rIBIS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General activities</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exploitation of past experiences</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recording of design rationale</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collaboration with other designers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handle communication and collaboration breaks.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collaboration with the business milieu</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>User involvement</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal statement</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goal elaboration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Sub)solution explication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agreement on solution</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2d-c: Synoptic form of the Software Design Support Framework (SDSF) – part 3
2.7 Main observations

The main observations on SDSF are as follows:

(a) There is a series of processes of creative/intellectual/human nature (see problem solving activity) for which there is no support whatsoever. This is probably an expected result.

(b) The present tools offer support for by-products of the design process i.e. design rationale and system knowledge, although Eiffel does explicitly deal with the representation of the design itself and its analysis should bring a balance to the framework in this respect. This means that there is a need to include more software engineering-based tools in the framework.

(c) In relation to (b), there is no tool so far that provides for the design itself as well as its by-products. As an example, LaSSIE acts as an information system which is completely separate from the computer system which facilitates.

(d) None of the included tools helps the designer to deal with needs of prospective users of the system and their tasks. This means that there should be included more tools that address issues of functionality and usability under the same scale and provide for activities like analysis and specification of task scenarios, conceptual and detailed interface design, design space analysis at the user interface level, dialogue specification and implementation, usability testing etc. In current practice, these activities probably take place independently - timewise - of the design and development of the rest of the software, mainly due to time/cost problems. Therefore the provision of this type of assistance by a tool must be flexible enough to enable the designers to use it at their will. It should not interfere with the rest of the design activities but co-exist with them.

2.8 Discussion

As far as the first observation mentioned above is concerned, the major issue is the applicability of machine involvement. In such tasks, designers usually assume that there is no machine support because this is where intuition and thinking comes in. Perhaps the assumption that "a machine cannot think, therefore I cannot rely on it to do the thinking for me" is not at place in this situation. What is tried is the assistance of thinking and intuition. Therefore what is desired is a set of computer-based support mechanisms that are available at hand without interrupting a designer's flow of thoughts and actions. On the contrary, I am looking for a type of support that will cause a design task to evolve. But how easy is it to help someone without getting in his/her way too much? Different attitudes should be dealt with differently, and so on.
The diversity of design activities between different projects and humans does not allow for a procedural assistance. It is too patronising for such tasks. On the other hand, a purely declarative approach of providing support to the designers merely as options on the screen is of questionable usefulness, because of creeping psychological obstacles like learning curves or cognitive overloads.

What is a surprisingly dominating phenomenon in software design, is the extraordinarily big number and diversity of the types of knowledge someone has to capture as well as the incredible jumps switches from a knowledge type to the other. If someone considers the representation issues for each knowledge type, this presumably poses a big amount of load to the designer. That is where expertise gets involved, providing knowledge (including metaknowledge) and skills to get through. If we only consider the information chunking capabilities of a human, then that point becomes even more obvious.

It seems that the representation issue is also quite important because.

(i) the diversity of knowledge types questions the adequacy of a single representation for all knowledge forms. On the other hand, multiple representations would raise problems of automatic transformation from one form to the other or the phenomenon of over-visualisation

(ii) the recognition that most of the learning during design is implicit, leads to the possibility that what is more important is the facilitation of learning as such and by any means, rather than the explicit representation of relevant knowledge. For example, working with a simulation of a mental model is enough to make a designer learn more about the system under construction, whereas the representation of this new knowledge in computer terms is obsolete. What is more important is the direct exploitation of this knowledge to refine the model, and produce something.

The switch between different learning situations/knowledge types is also important. A designer should be able to switch very easily and navigate through knowledge and different problem solving phases/abstractions meaningfully.

Another point to make is that the processes observed in Malhorta et al.'s participatory design session [62] are not much different form Guindon's process model of "software design for functionality" [37] which indicates that methodologies and tools for software design and user interface design could and should come together to avoid repetition of design tasks. A positive point of this framework is that it treats the design activities that aim for usability facilitation as any other software design process without distinction. This analysis of software design processes in contexts where there is direct user involvement, enables the identification of similarities between functionality- and usability-based design. More importantly, it allows user-centred processes to be spread in several places in the framework and thus allows for the
modelling of user involvement at different phases of the design process as it is believed it
should.

SDSF's realistic character reveals a view of the software design process which includes a
great deal of iteration of processes, constant experimentation, and no pre-defined route to the
solution. Therefore the aim of maintaining a realistic view of things is accomplished to a large
extent.

2.8.1 SDSF tools

In terms of the survey of tools, there's an inherent difficulty which has to do with the
ephemerity of the tools. New tools come out by the day and because most of them are
commercial and trying to keep up with the new ones, is difficult. Furthermore, because most of
them are commercial products, there is no demo version provided and the information we can
have on them is purely marketing rather than scientific. In the case where a scientific paper is
written that describes that package then things are fairly better. The strategy used to deal with
is problem is to: (a) select the tools that I think at this point in time that they are representative
of what I am looking for in my own judgement, (b) describe them given the available
information (c) plan the evolution of tool technology based on what I already know about them.
Future version of this framework might extend the list of tools into several classifications, e.g.
CASE tools, UIMSes, IPSEs, KBSes, etc..

Another point to be mentioned is that neither of these tools claims to be particularly
addressing any Psychological perspectives of the design process, it would thus seem unfair if
tools were judged in terms of their performance in that sector. In fact, the point I am trying to
make is that mainstream design assistant tools only consider software engineering theories of
design and do not take into consideration empirical evidence on the design process.

2.8.2 SDSF as a software process model?

In its current form, the software design support framework does not provide for a model of the
software design process but rather a summarising framework of software design activities
That is because:
(a) it does not contain any control information on how these processes are scheduled.
(b) it does not offer new empirical evidence of how software design is performed in practice,
but rather tries to (i) further analyse (or even formalise) existing evidence with further aim to
support some of these activities and (ii) include evidence from a series of diverse design
situation in order to provide a global view of design processes.

In the writer's opinion, the evolution of a new model would be a further task analysis and
normalisation of the framework's process structure.
2.9 SDSF refinement

2.9.1 Refinement of tool section

The first major change to SDSF is to abandon the "possible type of support" part. The reason is basically space. The information that column was presenting was at this point of time not directly useful, in the sense that at this stage, i.e., overview of the status of design support, one is not directly interested in technical details. At the implementation stage, one would be more interested on how specific processes are supported by certain vendors. However, at this stage it is considered better to save some space making SDSF more readable.

This space saving very easily led to further decomposition of the design subprocesses into elementary processes thus extending the number of process levels to three. The criterion for making a process elementary is the degree of ease with which one can determine whether the specific process is or is not implementable by a computer relatively easily. The process structure (i.e. "Design activities" part) is now considered to be more complete. A preview of the next version of SDSF is shown in figure (2h).

As mentioned earlier the use of a software design tool/environment would change the design context and it would be interesting to see how. A continuous cycle of empirical studies could then provide evidence for the improvement of such tools. Empirical evidence of task/user analysis and modelling sessions are also non-existent, to the writer's knowledge.

It was soon obvious that there was a problem with the current tools. The random selection of tools was not a good idea. Perhaps a different set of tools would do better on the observed weaknesses (paragraph 2.7). Trying to make the selection of tools more formal and efficient, the new approach involves the following stages.

(I) Indication: at this stage, a specific tool or set of tools is a candidate for inclusion in SDSF, if there are indications that it might do well in this context.

(II) Proof: at this stage it is decided to categorise tools in a similar way to the processes. Therefore there are three categorisations, each of which has about three categories. One typical tool is chosen for each category and it is included in SDSF. Judging by how well these typical tools do in the context of SDSF, certain categories are chosen for appropriateness.

(III) Adoption involves the isolation of the chosen tools/categories from the rest, which are now redundant.
<table>
<thead>
<tr>
<th>Design activities</th>
<th>Tool support</th>
<th>General activities</th>
<th>Subprocesses</th>
<th>Elementary processes</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>general → specific</td>
<td></td>
<td></td>
<td>Stage-oriented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Elaboration and understanding of the requirements</td>
<td>Inference of constraints</td>
<td>Consistency check</td>
<td>Meta-Edit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Handle uncertainty or breakdowns</td>
<td>Specialized schema process meta-schema</td>
<td>Issue prioritization</td>
<td>Analysis</td>
</tr>
<tr>
<td>Individual problem solving</td>
<td>Simulations in the problem domain</td>
<td>Multiple, sketch, retrieval simulation</td>
<td>Unplanned discovery, solution construct</td>
<td>Consist. check</td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Simulations of the design solution</td>
<td>Mental simulations, specific execution</td>
<td>External representation, design notation</td>
<td>Notation search</td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Formulation of solution models</td>
<td>Unplanned discovery, solution construct</td>
<td>Consist. check</td>
<td>Design notation</td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Solution development</td>
<td>Problem solving strategy, methods</td>
<td>Diagrammatic techniques</td>
<td>Navigation</td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Opportunistic behaviour</td>
<td>Relation of computer</td>
<td></td>
<td></td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Idea generation</td>
<td></td>
<td></td>
<td></td>
<td>Analysis</td>
</tr>
</tbody>
</table>

Figure 2e: The new version of SDSF.

2.9.2 Theoretical future work

Although it is obvious at this point which of the subprocesses in the framework receive little or no tool support (e.g. uncertainty and breakdowns, simulations, idea generation) it is not very easy to choose one of them for further research. That is firstly because there should be more tools included in the analysis, in order to give a more realistic character to the survey. Secondly, because some of these subprocesses are tricky in their support possibly because'
• it should first be adequately surveyed whether their support would improve the efficiency of software designers. For example, observation and interviews of users is best done by the designers themselves.
• the implementation of such a support may itself be difficult, e.g. some of the strategies employed in idea generation might be hard to implement.

On the basis of experimentation, the choice is made to try to support the series of breakdowns in software design on the grounds that:
• the number of events (i.e. elementary processes) that cause them is big, therefore the probability that these breakdowns will occur in design situations in the future is relatively high.
• the majority of the underlying elementary processes are implementable.
• individual elementary processes have been implemented by tools, but not more than two of them at once.
• it seems to be a key subprocess as it slows down the overall design process, and its alleviation will potentially smooth the process and subsequently offer support for neighbouring subprocesses like the formulation of solution models.
2.10 Breakdowns in the design and use of software systems

In this section, I describe a literature survey which specifically focused on breakdowns and neighbouring design activities. I first present the three different schools of thought on breakdowns as I can identify them, namely difficulties, conceptualisations, and mental model mismatches. The first two approaches refer to breakdowns as part of design activities and are the ones most interesting from the perspective of this thesis, whereas the third category refers to breakdowns in the context of software use. Within the first two categories we describe studies according to their relevance to this research, from distant to close. Then, purely for the sake of completeness, I present the studies that refer to breakdowns as mental model mismatches in chronological order, although we do not elaborate on these in much detail. Then, common ground among these types of breakdowns is identified and on that, a new role for them is built which has not been thoroughly investigated thus far.

2.10.1 Breakdowns as difficulties in designing software

Guindon

Guindon [35] reports on breakdowns as symptoms or causes of difficulties found by software designers while trying to solve a complex problem:

- Lack of relevant problem-specific design schemata
- Lack of or poor meta-schema for design
- Poor prioritisation of issues leading to poor selection from alternative solutions
- Difficulty in considering all the stated or inferred constraints in refining a solution
- Difficulty in performing complex mental simulations with many steps or with many test cases
- Difficulty in keeping track and returning to aspects of the problems whose solution refinements have been postponed
- Difficulty in expanding or merging partial solutions into a complete solution
- Too early commitment to an initial kernel based on a prion criterion (primary generator)

Guindon's seems to be the only trial to empirically define breakdowns in the context of early software design of a realistic problem. However, it mainly describes breakdowns as difficulties in the design process and does not stress at all any of their positive attributes referred to by the other authors mentioned in this study. He simply does not observe any of the positive recovery stages mentioned by the other authors.
Adelson and Soloway

Adelson and Soloway [1] refer to some sort of behaviour which resembles to Guindon's knowledge-related breakdowns [35]. Having an expert working on unfamiliar domain of expertise and on an unfamiliar object in that domain, they mention: "Here on expert had two oversights which were not detected because he could not us the domain specific skills which had prevented this kind of oversight when he was designing in his domain of expertise." The oversight refers to the designer not expanding certain functions to the extent he did with others.

I call this type of breakdowns type I breakdowns.

2.10.2 Breakdowns as conceptualisations

This view is attributed to the philosopher Heidegger. It refers to the process of recognition or identification of an object ("present-at-hand") after the realisation that this object is not readily available ("unreadiness-to-hand").

So there are two states for every object and its properties in the real world: "unreadiness-to-hand" and "readiness-to-hand". The transition between the former state to the latter state is attributed to the process of breaking down. It is therefore advocated that breaking down is the basis (one and only) mechanism for object emergence in our perception.

Literally speaking, breaking down is the reach of an "unreadiness-to-hand" stage. However, it should not be merely described as a "don't know what to do next" situation, i.e. lack of strategic knowledge, but it has more to do with focus of attention to an object in the rationalistic/traditional sense. For example, discovering new objects/structures through actions in everyday life is described as "trying to cope with breakdowns" but that's not either cognitively or psychologically difficult.
Winograd and Flores in [96] include a thorough discussion of breakdowns (BDs) in various contexts. Adopting Heidegger's viewpoint (referenced in [96]) they treat breakdowns as interruptions of our habitual "being in the world" that facilitate new understandings which are used to recover and continue with the task at hand. For example, when hammering a nail, the hammer as such emerges as an object only in the event of a problem e.g. the hammer breaks. Trying to fix it in order to continue with the task leads us to a new set of concepts and discoveries. This provides an interesting alternative view to the rationalistic perspective of breakdowns and problem solving in general which portray breakdowns more as difficulties or obstacles in the solving process (e.g. see section 2.10.1) ignoring any positive points in the recovery process. I call this type of breakdowns type II breakdowns.

![Diagram](image.png)

Figure 2f: Winograd and Flores's view of breakdowns as conceptualisations

Winograd and Flores provide the theoretical foundation for a new way of thinking about design describing breakdowns as a key activity through which learning and idea generation are facilitated.
Fischer [25] sees breakdowns as opportunities for creativity, which is close to Winograd and Flores's standpoint. The design process is highly dependent upon the tools we use, therefore breaking down is seen in that context. The following diagram outlines Fischer's view of a breakdown situation.

![Diagram of Fischer's view of breakdowns in design problem solving](image)

Figure 2g: Fischer's view of breakdowns in design problem solving

Building up on their previous work on providing critique to domain-dependent design environments, Fischer uses such computational environments to facilitate the occurrence of breakdowns by detecting and deviating from design rules/guidelines and letting the user-designer know and consequently tries to make the most out of it by providing critiquing information in the form of issue-based argumentation.

It is very interesting that Fischer's view of breakdown manipulation lies with argumentation-based tools as it matches my own view of the relationship between breakdowns and design rationale. However, Fischer differs in that he introduces the notion of design context which holds a central role in the whole breakdown cycle. In addition, Fischer argues that design rationale has to be augmented by rich context (by "providing assistance at the right time"). Fischer seems to be the only researcher who studies breakdowns in a tool-supported design situation, which is very important.
Fischer takes Winograd and Flores's theories a step further by extending them to (a) study several types of design situations and (b) providing tool assistance as the basis for the study of BDs. He also provides explicit computational platforms for the study of several types of BDs in several phases.

2.10.3 Breakdowns as mismatches of mental models

Bodker

In the context of Human-Computer Interaction, Bodker examines the role of breakdowns in computer use in general or even perhaps "computer task performance". Basing her work on human activity theory Bodker [5] also shares much of her ideas with Winograd and Flores's work. Nevertheless she takes their work a step further by describing a number of cases of breakdowns in computer use. She thoroughly analyses these breakdowns as well as their associated effects from her own theoretical viewpoint in interesting ways.

In a theoretical vein, some of the positive outcomes of breakdowns are conceptualisation and openings for learning which are elaborated in [5] as follows: "The individual human being possesses a certain repertoire of operations. This repertoire forms the basis from which operations are triggered when met by concrete, material conditions. For each concrete action, the human being is dependent on the triggering of a sequence of operations. Actions can be turned into operations; operations can be conceptualised. The operations applied in a specific action are not conscious to the human being. But through conceptualisation, they can be made conscious to us as the actions they once were, we can name a specific sequence of operations and understand and explain reasons for their application at the level of the former actions. Conceptualisation can take place in breakdown situations wherein some unarticulated conflict occurs between the assumed conditions for the operations on the one side and the actual conditions on the other".

Bodker also follows Winograd and Flores's view of breakdowns. She treats BDs as unintentional shifts of focus bringing new objects etc. to the attention of the designers. However, she does not provide any software design examples.
Jambon

Jambon [47] works on a similar issue which is task interruption. This can be thought of as unintentional focus shift caused from external 'sources'. He/she approaches the issue rather practically, building a model of task interruptions as well as a formal notation that best represents that model. The notation is to be tested in a real-world system. The main emphasis is at the task rather than the cognitive level.

Krasner

Krasner et al. [54] slightly differ from the rest of the authors cited here in that they talk about communication breakdowns in large software teams. Although this is an interesting extension to the term 'breakdown', it is not of explicit concern in this research work.

Sharples

Sharples [80] uses the term breakdowns to describe both trouble found by users to use a prototype system as well as communication obstacles between peer students while using that prototype.

2.10.4 Discussion

Perhaps Guindon refers to breakdowns in the same way as Heidegger and Winograd (see figure (2k)) but the latter being so general and philosophical, - quote for example "it's only meaningful to talk about the existence of objects/properties when there's potential for breaking down (concernful activity)" - they have the ability to characterise breakdowns in a number of ways either positive or negative. On the contrary, Guindon by restricting the subject into an individual software designer's actions when solving an ill-defined problem is bound to identify a series of breakdowns which are clearly negative, in the sense that their alleviation is not an easy job for the designer.

![Guindon diagram](image)

Figure 2h: A collective view of breakdown theories.
It is very interesting that most of the above studies share the view that undetected or unresolved breakdowns in the design process are transformed in the design product and may reveal themselves as problems in its use. In fact, breakdowns seem to have a different face depending on the phase in the life cycle of the product in which they are made explicit. In this respect, it seems obvious that the earlier they are detected the better.

From the above studies, I have chosen Guindon's and Fischer's as the main points of reference. Guindon's work is the closest to this work's domain of interest, i.e. high-level software design - the domain is very important in the understanding of human design activity. Fischer's work is very important among other reasons because it deals exclusively with tool-supported design and thus sets the scene for a modern context for tomorrow's designer. Jambon's work is also very interesting as it demonstrates the application of theoretical behavioural models to industrial practice; I did something similar in the SEDRES project described in chapter 3.

2.11 The prospect of theories of argumentative reasoning with process support

Breakdowns as decision making or reasoning points

In this study the relationship between different types of breakdowns (mainly Guindon's) and designer's reasoning and decision making is investigated. It is felt that breakdowns points are key points in the designer's planning and decision making process in terms of the overall design progression. The theory is that by assisting the decision making or reasoning process, breakdown alleviation/anticipation/generation can be optimised. Hints have been made by Winograd and Flores [96] and MacLean et al. [61].

The idea is to base the solving route of software design on design rationale. Many notations exist for representing design rationale, e.g. the one described in [61]. The adoption of such a technique and its extension in the form of a working prototype could reveal a lot about software design. Some arguments in favour of this approach are:

• The initial assumption that software design is a problem-solving activity, and the characteristics of an ill-structured problem [40], provide a suitable theoretical basis.

• Design rationale inherently combines domain and solution knowledge; therefore provides a suitable means of combining problem/solution (or analysis/design) models.

• Provides a good representation basis for navigation through design processes.

So the work is moving towards a network-based system whose main nodes represent design issues and adjacent nodes represent options, criteria and decisions, very much in the same way as many such tools base their models. As an extension, each node could also include
data/knowledge of relevance to the issue at hand, such as design notations, constraints, heuristics etc. Different media could be used as appropriate to represent different design process elements. Arguments in favour of such a system, are:

- Such a system could provide guided exploration of the design space. Also development/expandings of the solution: decomposition, appliance of methodological operators, problem-solving strategies etc.
- Facilitation of implicit learning.
- The integration of design rationale with the rest of the design components, which does not exist in current tools.
- Such an approach changes the focus (key) of navigation from design notations (as is the case in most commercial tools) to design argumentation. It could also let designers decide on the navigation key by themselves. Subsequently, another potential improvement is that such a tool could extend navigation to a higher level, right at the top of the process, as it can provide a unified way to deal with argumentation/justification throughout the system lifecycle.

Most of the above arguments are positive in terms of breakdown alleviation. However, it must be remembered that most of them are simply hypothetical at the moment. A potentially negative point of such a system could be the initial impact on designers' speed. That is because people think faster than they act. Therefore especially in simple argumentation issues, the use of a computer could prove to be a slowing down factor.

However distant the points of emphasis between type I and type II breakdowns may seem, it looks like a common piece of ground in all types of breakdowns is at the causal stage (see figure (2k)) the lack of readily available knowledge of strategies that correspond to the particular problem class being worked on, as well as the discovery of such knowledge, at the recovery stage. The issues that are raised are: (a) what can be done about breakdowns? Can we interfere to ensure a productive recovery?, (b) In a design context, how can I transfer knowledge across designers in a way that recovery is facilitated? Design rationale has been suggested as an appropriate way to do that.
2.12 Chapter summary

In this chapter, I have investigated the software design process looking for interesting points in it on which tool support could be improved. Literature survey has led to breakdowns in designers' behaviour as key processes as many actions seem to be related to them and they form the type of process that is least well supported by software tools. Looking deeper into the breakdown literature, a set of different types of them is found. Despite their differences, common themes emerge which focus on the knowledge-rich aspects of uncertainties and oversights as well as strategic difficulties. Inspired by Schon's theories on reflection in action, I suspect that decision making is crucial to optimal breakdown recovery and pursuit that further by studying the current design rationale methods and tools. Before that though, more needs to be learned about design processes from own experience. As most of the work so far is based on the available bibliography, it is felt that is essential that the survey is complemented with my own study of the process. The first step in that direction is to study design practice at the industrial setting. The SEDRES project described in the next chapter provided us with such an opportunity.
PART B: Understanding and Modeling Design Practice

CHAPTER 3

THE SEDRES PROJECT

Table of contents

3.1 INTRODUCTION ................................................................. 63
  3.1.1 PROJECT SUMMARY ..................................................... 63
    3.1.1.1 OVERVIEW ........................................................... 63
    3.1.1.2 AIMS ................................................................. 63
    3.1.1.3 OBJECTIVES ....................................................... 63
    3.1.1.4 THE PROBLEM CONTEXT ........................................ 64
    3.1.1.5 RELATED WORK .................................................. 64
    3.1.1.6 FURTHER INFORMATION ....................................... 65
  3.1.2 THE ROLE OF SED RES IN THE THESIS .............................. 65
  3.2 WORK PACKAGE 3 (WP3): 'DEFINE PROCESS' ............................ 66
    3.2.1 OBJECTIVE ........................................................... 66
    3.2.2 APPROACH ........................................................... 66
    3.2.3 TASKS 3.1, 3.3 ....................................................... 66
      3.2.3.1 OBJECTIVES .................................................... 66
      3.2.3.2 APPROACH ...................................................... 66
      3.2.3.3 DESIGNERS INTERVIEWS .................................... 68
      3.2.3.4 SURVEYS: DISCUSSIONS AND FEEDBACK FORMS ......... 69
      3.2.3.5 DEVELOPMENT OF A HUMAN-ORIENTED HYBRID NOTATION FOR THE
               SYSTEMS ENGINEERING PROCESS ................................ 70
    3.2.6 THE SEDRES GENERIC PROCESS MODEL (SGPM) ..................... 74
    3.2.7 HARMONISING PARTNERS MODELS W.R.T. THE GPM .............. 78
    3.2.8 CONCLUSIONS AND RELATIONSHIPS TO SUBSEQUENT TASKS (T3.2, T3.3) 82
  3.3 WORK PACKAGE 6 (WP6): 'DEMONSTRATE WITH REPRESENTATIVE MECHANISMS' 86
    3.3.1 OBJECTIVE ........................................................... 86
    3.3.2 GENERAL APPROACH .............................................. 86
    3.3.3 DATA COLLECTION BY OBSERVATION (THEORY AND PRACTICE) ..... 87
    3.3.3.1 WHAT IS EVALUATED ........................................... 87
    3.3.3.2 MONITORING OF FORMS ......................................... 87
    3.3.3.3 OBSERVATION ..................................................... 88
    3.3.3.4 OVERVIEW OF DATA COLLECTION TECHNIQUES .................. 90
    3.3.4 THE NU*DIST SOFTWARE AND ITS ROLE IN THE EVALUATION EXERCISE 91
    3.3.5 EVALUATION PROCEDURE (BY EXAMPLE) ............................ 92
3.1 Introduction

This is the first of two studies which gave us a better understanding of design practice. Work on the SEDRES project enabled us to perform what could be close to a field study of design practice in the aerospace industry. A host of information on designers at work using software tools was gathered, and many practical problems were solved that relate to modelling the systems engineering process as well as evaluating design work. Certain terminology – specific to the project – is sporadically used in this chapter. A glossary of terms is included in appendix A. Its purpose is to offer clarification of the exact meaning of those terms.

3.1.1 Project summary

3.1.1.1 Overview

SEDRES is a CEC Framework funded ESPRIT project that started in January 1996. It involves the systems organisations within the following aerospace companies: Aerospatiale, Alenia, BAe MAD, DASA, SAAB. In addition two universities are also involved, Loughborough in the U.K., and Linkoping in Sweden. The project is due to run for three years, and involves over 40 person years of effort. More information on this project can be found in [79] and [88].

3.1.1.2 Aims

The project is directed at the development of an interface standard which will allow the complete set of tools in design of airplanes and spacecrafts and their avionics systems (hardware, software, mechanical design, and implementation, project management, etc) to communicate in such a way that an integrated project support environment is available from commercial tools. The interface standard will possibly be based on the STEP interface used in CAD applications, but will be extended to add the semantics appropriate to the aircraft industry.

3.1.1.3 Objectives

This project will develop and demonstrate an 'open' design information environment for the active components (e.g. electronics, hydraulics, computing, etc) of complex integrated products. The resulting standards and environment will increase the efficiency of industrial collaboration in design and product support, and will complement the strong focus on geometry and structure definition which characterise other research projects, such as those under AIT.
3.1.1.4 The problem context

A significant part of all complex, integrated products (such as aircraft, vehicles, transport systems and even buildings) is systems. In an aerospace context, aircraft systems comprise: Avionic systems (mission, communications, navigation, human/systems interface etc.), and Airframe systems (crew/passenger escape, power generation/distribution, environment control, fuel management, etc.) A "system" in this context is defined as a set of predominantly active components, many of which are electronic components like embedded computers, sensors, displays and actuators, that are interconnected via dedicated direct links or by communications buses. Consequently, "systems" cannot be simply considered as a sum of the single components; rather, their integrated behaviour, which in most cases is real-time dependent, has to be defined, validated and verified. In typical contemporary systems, there are tens or hundreds of interacting components, resulting in an enormous degree of complexity.

An increasing number of design tools are used, covering all aspects of such systems, including types of architecture and product behaviour. This situation leads to increased information duplication, inconsistencies, checking and correction. Also there is little consistency to the configuration management approach implemented across all tools. Since the design information is captured within the tools, it is in a proprietary format, and this leads to problems, such as increasing dependency on tool vendors and the tools acting as a barrier to co-operative working due to the investment necessary to move to common tool-sets. This project will clarify this problem area, define requirements for, develop and demonstrate both:
(1) data exchange & storage standards and
(2) an example of a comprehensive systems design environment based on commercially available tools, that will address and solve these issues.

3.1.1.5 Related work

Exchange standards have provided candidate PDES/STEP Application Protocols for the systems elements of product definition (behaviour, timing, non-functional properties). The project work is done against the background of the deficiencies of current Systems Development frameworks (for instance, DoD 2167A, 499B, EIA IS 632, IEEE P1220, etc.), and is therefore establishing a basic Concurrent Engineering Concept, a union of process, development environment, tools and engineering techniques against which the key objectives of the project are achieved.
3.1.1.6 Further information

The Project involves contributions from Systems Developers (as IT-Users) providing the key problem areas and business pragmatics, from CAE/CASE tool suppliers (as IT-vendors) providing solution technologies and tool development pragmatics, and from research institutions, to underpin approaches with sound theoretical foundations. The project also builds on the work of the structures/airframe parts of organisations, by extending the work on the Digital Product Definition (DPD), into becoming a Comprehensive Product Definition (CPD).

The exploitation, in general, aims to improve quality and reduce cost & timescales for systems development, and the exploitation process will firstly apply the results internally to current product developments, secondly, spread the results to external organisations, such as the partner companies on large projects and thirdly, the companies will follow an active policy of dissemination of the public results of the project, for instance, in promoting ISO/STEP standards, in encouraging tool suppliers in providing STEP-compliant interfaces, and in promoting education of data exchange to support concurrent engineering in their educational policies.

3.1.2. The role of SEDRES in the thesis

The best thing about the SEDRES project is that it gave us direct access to design practice in ‘the real world’. Although the systems engineering theme prevents us from seeing things as ‘strictly software’, it is a fact that most of the industrial partners are currently or have been in the past active software engineers or software project managers. This gives them the right influence from the software development arena. In addition, I can count on the similarities of disparate engineering disciplines, including software engineering. That very essence is one of the main assumptions in the field of Systems Engineering.

On the one hand, I introduced to the representatives of the industrial community of alternative (human process-based) ways of thinking about design processes. On the other hand, there has been ample opportunity to test theoretical/empirical models in practice.

The author’s involvement in the project spanned two periods: January to May 1996 and February to November 1998, covering parts of Work Package 3 (WP3) and Work Package 6 (WP6), respectively. WP3 is a good starting point of capturing design practice as it caters for retrospective process modelling at the higher scale. WP6 complements the picture as it involves evaluation work of the SEDRES Data Exchange Mechanism (SDEM). That involves observation records as well as informal communications within engineering work practice.
3.2 Work Package 3 (WP3): ‘Define Process’

3.2.1 Objective

The focus of this work package is to formulate the minimum feasible development process model of the concurrent engineering design processes, against which the configuration management and tool design data exchange needs will be formulated. This process model will also be the process which is trailed in the later demonstration phases, where there must be a clear definition of the way parallel engineering work ties together.

3.2.2 Approach

From the definitions of the use-scenarios, formulate a definition of the practical use process to be used, identifying the roles of organisations and design tools in that process, and of the minimum formal configuration management model to be used. Apply a systematic test & evaluation approach to validate that these definitions are adequate for their declared roles.

3.2.3 Tasks 3.1, 3.3

3.2.3.1 Objectives

In Task 3.1, the objective is to define a process framework against which the different process models for the different use scenarios can be built against.

In Task 3.3, the objective is to allocate roles of people and tools to the specific processes of the use process model derived from the generic process model of Task 3.1.

3.2.3.2 Approach

Task 3.1 performs two activities. Firstly, the definition of each company’s individual use-process (the common process that underlies all the use-scenario work) will be against a backdrop of a Generic Process, something akin to the conventional ‘waterfall’ development lifecycle. The Generic Process will provide a common full-lifecycle reference, against which the processes of the individual use-scenario processes can be placed.

Secondly, it is necessary for a particular convention and notation to be used to express companies’ own process models; this activity is meant to propose the process model style and notation used for subsequently process model definitions.

In Task 3.3, the definition of a process to take a given set of input requirements (such as the requirements for part of an aircraft and its systems) and produce a given set of outputs (such as architectural descriptions, technology definitions, acceptance criteria, system evaluations...
and test results) comprises: the activities that must be performed in the process, any intermediate products, and the casual and information dependencies between these activities. These activities are performed by a combination of engineers, performing the value-added elements of the design activity, and design and management tools supporting the mechanistic operations (data presentation and conversion, configuration management, document generation, etc.).

This task is meant to take the definition of the process from previous activities, and allocate activities to engineers and tools, in a way which maximises the effectiveness of both the process and of the strengths and capabilities of people and machines.

Two of the companies involved in SEDRES had already provided us with their "official" systems engineering development process models. It was obvious that these were very similar to traditional software engineering-based life cycle models like the waterfall model etc. The aim was to include (a) the aspect of concurrency currently missing and (b) aspects of human behaviour as well as involvement of computer-based 'tools' in the process, especially with respect to Task 3.3.

Another obvious point was that these process models were also similar to industry standards such as the IEEE P1220 [43]. Culturally, none of the companies wanted something radically new which would be fully contrary to their familiar models. Therefore the new model should compromise 'standardness' and realism in one piece.

As a first major step in deriving part of the Generic Process Model (GPM) out of design practice, a series of interviews was conducted with systems engineers and project managers. The interview templates were originally designed in previous LUTCHI projects and tailored to the SEDRES realm by the author, who also conducted the interviews themselves. Nevertheless, it has to be said that prior to the interviews there was already a fair amount of understanding with respect to design practice, through the following:

- Presentations of the engineering design practice at different levels in BAe.
- Project technical meetings where exchange obstacles were discussed from the viewpoint of process management.
- Informal discussions at the side of those meetings where many of the issues raised in a meeting would be taken further and people would converse more freely about the exchange problems and their potential solutions.
3.2.3.3 Designers Interviews

A software engineer, a software project manager and a systems engineer out of two of the companies were interviewed in order to establish a more concrete picture of design work in the aerospace industry. Interviews lasted about an hour and progressed around three focal themes: (I) The overall systems engineering process ('big picture'), (II) The systems engineering process in the context of Use Scenario 1 and (III) Computer system and human support to the process. The structure of the interviews is presented in full in Appendix A.

When referring to the design process interviewees to a greater or lesser extent advocated their companies' 'official' models. However, there were quite a few points where it was made clear that the current way of working was slow and inefficient. One example is the procedure of providing feedback from the detail design team/level to the conceptual design team/level. Although the BAE process model suggests that Detailed Design clearly precedes Detail Design, in practice these two types of activities do take place in parallel for some time. The exchange of information between the two teams currently happens timing - pieces of paper placed into pigeon holes awaiting for response.

In fact, overlap across design stages seems to be inevitable in a real project. What is interesting, though, is that there is no measure of objective or firm criteria on under what conditions the higher design phase has to freeze in order for the implementation to carry on regardless (by itself). At the concurrency period, information exchange is not too systematic. These problems are outlined in the following diagram drawn in an informal meeting with one of the industrial delegates.
3.2.3.4 Surveys: discussions and feedback forms

At the stage where the initial thoughts of a SEDRES Generic Process Model (SGPM) were in place, I wanted to test those thoughts with the interested parts and also get some more concrete feedback on what the expectations were. To do that, I performed a survey. The idea of surveys was initiated within LUTCHI and designed and carried out by the author.

The approach at this point was to start off from a nominal model. In a meeting at BAe, I presented to the BAe people a set of alternative models that had been used in similar cases in the past. These include the MULTIK model, the Star model [41] etc. We discussed the 'pro's and 'anti's of these models, getting a first feel of the partners intentions and deliberations. It had been suggested by partners that the IEEE P1220 is a good starting point, as it is a widely accepted industry standard.

At this point the objective is to get to know more about attributes of the model partners would want to see more explicitly.
Feedback forms were circulated among attendees in the 4th Technical meeting after a presentation of these initial thoughts in the form of a model skeleton. Partners filled the forms and these were collected at the end of the second day of the meeting.

The following paragraphs illustrate the SEDRES Generic Process Model (GPM) as well as its notation. Both of these were created solely by the author. For the sake of a better illustration, the notation is presented first. Layouts of the feedback forms are included in appendix A.

3.2.3.5 Development of a human-oriented hybrid notation for the systems engineering process

General overview

The two requirements that are derived from the aims (see section 3.2.1) of the generic systems engineering model – often referred to in this document as GPM – are:

(a) that it can be sufficiently clear and intuitive so as to be able to support communication between the SEDRES partners and between the SEDRES project and the world-wide systems engineering community;

(b) that it has a sufficiently comprehensive notation which can represent both a generic systems engineering model, and the aspects of process that see in (i) company specific systems development processes and (ii) the particular subsets of processes that we see in the Use Scenarios.

With respect to the notation, this is how these requirements are implemented:

As far as (a) is concerned, the project agreed that the GPM is compatible with the IEEE standard 1220-1994. Notationally, this means that the GPM should provide a suitable representation of that standard.

As far as (b) is concerned, a notational link must be established in order to connect the generic with the specific models. The experience so far shows that functional thinking is dominant among the industrial partners. Consequently, the "functional perspective" becomes that link and serves as a notational platform. Therefore this model describes the system development process from a functional perspective, i.e. in terms of functions and their interactions as data/information flows. The notation is a blend of existing and well established notations, slightly modified or extended to accommodate the needs of SEDRES as required.
The notation has been defined for internal project use. An early task in the second part of the project will be to consider compatibility with the STEP standard. Thus a revised version is to be expected.

**Purpose**

A list of attributes that the generic process model should exhibit is listed below:

- readable
- relatively simple
- intuitive
- informative
- flexible
- representative of the current engineering process

Although there are cases where one might see some conflict between some of these attributes, the purpose of the notation is to enable the generic process model to exhibit or accommodate most of these requirements.

**Description**

The main point about this notation is that there are three viewpoints from which the systems engineering process is described: (full product) life cycle, (elementary) systems engineering and Use Scenario. The first two viewpoints correspond to the levels of abstraction that one can identify in the development process of the products in the aerospace industry. The third viewpoint is specific to a certain system which could be at any abstraction level and provides a refinement of the current processes in order to analyse a specific case study. Consequently, there are three types of processes/functions: life cycle processes, systems engineering processes and Use Scenario processes. This is made explicit in the notation, so that the reader is able by first look to identify which level they are in and also how the adjacent level processes are related. More information on this categorisation of processes is provided in subsequent paragraphs.

Another important point that has to be made clear, is that there are three different and complementary representations which are used. Each of these representations is implied by a set of notational objects.

The first representation refers to a product structural diagram, and uses rectangles and continuous lines (notational objects #1,2) to represent system architectures.
The second representation refers to the functional perspective of the process at different abstraction levels and uses the rest of the notational objects apart from the rectangle bar (notational object #7) to represent processes, data flows, etc.

The third representation refers to the timeline view of the process and is basically a Gantt-type chart which uses the rectangle bar (notational object #7) to represent process durations and sequences.

The notation is shown in figure (3b) and its associated tutorial is included in appendix A.
Notational objects

(1) □ : system architectural element
(2) ---- : "consists of" relationship
(3) ○ : life cycle process
(4) ○ : systems engineering process
(5) ○ : Use Scenario process
(6) ---- : data store
(7) ---- : process duration
(8) ---- : data/information flow
(9) ---- : designated area (e.g., process area)/categorisation
(10) * : multiple instances (notational object repeated n times, n>0)

Note: T3.3 specific notation

Notational objects

(8a) ---- : electronic design data flow
(8b) ---- : electronic flow of additional information
(8c) ---- : physical information flow

Actors

(1) ○ : computer tool
(2) ○ : human using tool
(3) ○ : human individual
(4) ○ : team

Figure 3b: The SEDRES Generic Process Model notation
3.2.3.6 The SEDRES Generic Process Model (SGPM)

**Background information**

As explained above, the "generic" portion of the model is derived mainly from P1220 [43]. The generic systems engineering process is adopted as is, with some changes on the shapes that are used, in order to provide the right connections and compatibility with the rest of the notation. The life-cycle processes provide a visually richer form of the descriptions given in P1220. In addition, they accommodate the basic features of a V-type life-cycle as requested by the industrial partners.

Another list of notions, e.g. iteration, concurrency, process areas etc. are adopted from the CMM [70]. The blend of these two well-established standards ensures the facilitation of the model with some of its most essential attributes, which are readability and ease of communication. The rest of the notation is derived from standard texts like Yourdon's etc. [97]. Again, this serves the easy recognition of the model. The T3 3 specific notation is initiated by the ideas presented by Sutcliffe and his colleagues in [87].

**Description**

The model is shown in the following figures. The first part of the model shown in figure (3c) describes the systems engineering lifecycle at an abstract level. The main objective is to provide a generic view of the overall process that can fit in different companies and situations. It includes a functional perspective which describes data flows and functions, and a system architectural perspective which shows the corresponding physical system structure.
Figure 3c: Generic systems engineering lifecycle

Figure (3d) shows an instantiation of this model that takes into consideration software design/development, as is very much the case in SEDRES.

Figure 3d. Systems engineering lifecycle including the software process
Figure (3e) shows the generic systems engineering process. An instance of such a process is expected to take place at different phases of a project life-cycle. The exact form of that instance is determined by the particular company that performs the process, the stage of the life-cycle etc.

Figure (3f) illustrates the systems engineering processes at the level of system definition, subsystem preliminary and detail design. It also provides a generic process model but this time at a specific phase of the life-cycle which is more related to the SEDRES scope. It also highlights the area which a potential Use Scenario fits into in terms of process.
Figure 3f. Placement of Use Processes in the context of the SGPM
Limitations

(a) As mentioned earlier in this document, the SEDRES Generic Systems Engineering Process Model in its current form concentrates on the functional perspective of systems engineering. Perspectives that are not represented at the moment include data management, control flow, organisational etc.

(b) The representation of process concurrency is limited.

3.2.3.7 Harmonising partners models w.r.t. the GPM

As the SEDRES Generic Process Model - and especially the corresponding notation - was meant to be used throughout the rest of the project, its adoption by the involved companies was very important. Given the strong feelings the companies have for their official process models, certain concessions had to be made from their part in order for the generic model to be adopted to its full. At the relevant technical meeting, it was decided to initiate a model harmonisation process, through model cross-reference. That task was carried out by the author.

Cross reference of company process models

This section provides a cross-reference table which shows how the individual process models fit on the generic process model. The focus is on the first two levels of abstraction, i.e. lifecycle and elementary systems engineering, and it can be seen how generic processes are supported at an individual company level and how the various models differ in terms of content and terminology. The different company models are included in appendix A.

Table (3a) enables process cross reference. The purpose of this table is not to “compare” (in its strict sense) the different company models, but rather to facilitate the understanding of the individual models and thus provide the feedback towards a SEDRES-representative generic process model. It must also be noted that the table does not intend to exhibit any form of process sequence or iteration. The intention is to provide a descriptive view of the terminology that is used through the companies.
The cross-reference table is comprised of two parts: the generic part and the companies part. The generic part provides a textual description of the generic process model (GPM). In particular, it includes the first two types of processes i.e. life-cycle and systems engineering.

The companies part includes the corresponding stages and terms that are used by the individual SEDRES companies, with each company occupying one column in the table. At the moment, I have represented an idealised situation where a system consists of a number of sub-systems and each sub-system consists of a number of components. However, in practice, different companies deal with different system sizes and structures (see appendix A). For example, a company may be developing a system which contains a number of levels.

Table 3a: SEDRES process cross-reference table

<table>
<thead>
<tr>
<th>Life cycle process</th>
<th>Systems Engineering process</th>
<th>Aerospace</th>
<th>Alenia</th>
<th>BAE</th>
<th>DASA</th>
<th>SAAB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirements Analysis</td>
<td>System Requirements Analysis</td>
<td>System Requirements Analysis</td>
<td>Requirements Analysis</td>
<td>System Concept</td>
<td>Overall Airborne System Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Functional Analysis</td>
<td>System Functional Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Functional Verification</td>
<td>System Functional Verification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synthesis</td>
<td></td>
<td></td>
<td></td>
<td>Proposed System Architecture</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Physical Verification</td>
<td></td>
<td></td>
<td>System Functional Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sabsystem Preliminary Design</td>
<td></td>
<td></td>
<td>Subsystem Requirements Analysis</td>
<td>Subsystem Functional Analysis</td>
<td>Sub-system Design</td>
<td>Equipment Functional Design</td>
</tr>
<tr>
<td></td>
<td>Requirements Analysis</td>
<td>Subsystem Requirements Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requirements Baseline Validation</td>
<td>Subsystem Requirements Validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functional Analysis</td>
<td>Subsystem Functional Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functional Verification</td>
<td>Subsystem Functional Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synthesis</td>
<td></td>
<td>Subsystem Synthesis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical Verification</td>
<td></td>
<td>Subsystem Physical Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sub-system Detail Design</td>
<td></td>
<td></td>
<td></td>
<td>PC</td>
<td>LRI Design</td>
<td>Software/Equipment Architectural Design &amp; Software/Equipment Development</td>
</tr>
<tr>
<td></td>
<td>Requirements Analysis</td>
<td></td>
<td></td>
<td></td>
<td>Software/Equipment Requirements Analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requirements Baseline Validation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functional Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functional Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synthesis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical Verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NA: Not Applicable. PC: Fully corresponding with the generic process model/ PI220.
of subsystems before these are broken down into components. In addition, different companies use different terms for their products. This is exactly the aim of the cross-reference table, that is to facilitate discussion among partners in order to form a single paradigm of system structure and terminology.

**Correspondence of company process models with GPM**

In defining the process of representing the different company models using a single process model, the correspondence of these models with the SEDRES GPM appears in three distinct ways:

"LEVEL 1 - notation" means that all companies agree to use the same notation, i.e. the set of notational objects proposed in W31.1.

"LEVEL 2 - process decomposition" means that all industrial partners agree to the existence of same structuring of processes. The two main rival approaches (and probably the only applicable ones in this case) are (a) organisation of processes according to system architectural structure and (b) process organisation in terms of system development stages. Due to the different types of products produced by the industrial partners, the organisation of the GPM in terms of the system development stages has been taken as the dominant structuring approach.

"LEVEL 3 - full correspondence" means that all companies agree:

(i) on a common system structure and
(ii) on a common terminology

The cross-reference table provided in this section serves exactly that purpose, i.e. to exhibit the progress done in Levels 1, 2 and facilitate the two Level 3 stages.

If we go back to the two main aims of the GPM (paragraph 3.2.1), then the first aim (i.e. the service of internal and external communication) needs correspondence for Levels 1,2 to be accomplished. The second aim (i.e. "placement" of the different company models on a single process framework) can even be accomplished at Level 2 providing that a common system structure (Level 3/(i)) is agreed at this level, instead of Level 3. The progress of T3.2, T2.4 needs agreement on Level 3.
Observations on the cross-reference table

- Aerospatiale's generic process model corresponds very much with the P1220 standard

- The process models of BAe, DASA and SAAB do not represent the systems engineering processes of Synthesis and Physical Verification at the System Definition level, whereas in Aerospatiale's model they are termed System Physical Analysis and System Physical Verification, and in Alenia's model they are merged in a single systems engineering process, termed Propose System Architecture.

- At the System Definition level, the first four systems engineering processes, i.e. Requirements Analysis, Requirements Baseline Validation, Functional Analysis and Functional Verification are treated differently in the different company models. In Alenia and SAAB they take place as a single process termed System Requirements Analysis and Overall Airborne System Design., respectively. In the BAe model they are represented as two processes, i.e. Requirements Analysis and Functional Analysis. In the DASA model they are represented as System Concepts and System Requirements Analysis, and in the Aerospatiale model as three processes, namely System Requirements Analysis, System Functional Analysis and System Functional Verification.

- Similarly, at the Subsystem Preliminary Design level, out of the six systems engineering processes included in the GPM, the Aerospatiale model corresponds fully with P1220. The Alenia model proposes three corresponding processes emphasizing the distinction between requirements analysis, functional requirements analysis (design) and (physical) system architecture. In the BAe and DASA models there is one corresponding process i.e. Subsystem Functional Analysis, and Subsystem Requirements Analysis, respectively. In the SAAB model, there are three processes: Subsystem Design, Equipment Functional Design and Detailed Functional Design.

This mapping has clearly highlighted differences in interpretation and understanding of the GPM particularly with respect to Alenia, BAe and DASA. These three industrial partners have provided different cross references to the GPM despite using company models derived from the same aircraft project (EF2000).

These differences appear to be due mainly to, different interpretations of the GPM and the use of different levels of abstraction within the company processes. For example each of the partner companies has a different interpretation of 'Synthesis' and 'physical verification' for the GPM 'Subsystem Preliminary design stage':

BAe: The functions defined in this design stage (Subsystem Functional Analysis) are mapped to specific physical components. This mapping of the functions to components has been interpreted by BAe as being equivalent to 'synthesis' and 'physical verification' of the
GPM. BAe have not made explicit reference to the development of a physical architecture as a separate 'subprocess'.

Alenia: 'Synthesis' and 'Physical verification' have been interpreted by Alenia to represent a separate design stage of 'Revise System Architecture'.

DASA: No interpretation of 'Synthesis' and 'Physical verification' has been provided by DASA which may be due to the physical mapping of functionality within this design stage not being explicitly defined in the theoretical process model for the project.

The Aerospatiale cross reference appears to follow the GPM notation very closely, however it is important for further discussion to take place between the SEDRES partner companies to confirm that the GPM explanation of the subprocesses (defined at task level by IEEE-1220) is equivalent to the Aerospatiale definition of the terms used in their organisational process.

The most difficulty was found in interpreting the SAAB model against the GPM. SAAB have produced a model which very roughly cross references to the GPM but the notation is different from the other industrial partner inputs. It would therefore be valuable to discuss the SAAB interpretation with all the partner companies in order to confirm that their understanding of the GPM is consistent with the other partner companies.

This Cross Reference table has highlighted where the GPM model has been interpreted inconsistently and now provides a reference point for resolving different interpretations and improving overall understanding of this common model.

3.2.3.8 Conclusions and relationships to subsequent tasks (T3.2, T3.3)

General discussion

The generic process model shows the first attempt to represent the systems engineering process in the context of the SEDRES partner companies. At the moment, the model describes the involved process from a functional perspective. Therefore, it combines (a) a viewpoint familiar to the internal and external engineering community and (b) a series of well known standards and notations like P1220 [43]. More perspectives could be accommodated in later project stages.

It is envisaged that such a generic model/notation can facilitate later project tasks that involve process modelling of some sort. What is further required is a common agreement between the industrial partners. This agreement is seen not only in terms of notation or process decomposition, but also as a common system structure and a common terminology.

It is intended that the GPM will provide a common language for each of the partners involved in the project. In order to achieve this objective it is essential that all the partner companies now discuss their understanding of the process and particularly to distinguish differences in partner company processes from where differences exist due to inconsistent interpretations of
the GPM. Initial experience from the project has shown that it is essential to develop a generic model of the design process to provide a common framework and language for communicating across all the partner companies and to clearly understand process differences between each of the three Use Scenarios.

It should be noted, however, that any further discussion centred around process has to be performed against, and potentially limited by, the key SEDRES objective - the development of a workable data exchange standard.

**Relationship to Task 3.2**

One of the objectives of the GPM is to provide a suitable notational platform that will be used to represent processes in subsequent project tasks. In Task 3.2, the notation must be used to represent the development process activities that are seen in the different use scenarios as well as how these interact etc.

Figure (3g) shows how Use Scenario1 (as an example) can be represented functionally without losing the context of the overall systems engineering process. Figure (3h) shows a timeline view of the same process structure. At this abstraction level, we are not talking about a generic model, but of a specific application process.

It is also worth noting that there is a need in figure (3g) to separate the actual Use Scenario design work from the work which is performed to provide realistic boundaries. At this stage, this need can be accommodated by using the designated area mark (dotted line) to show which of the Use Scenario functions are representative, and so forth. At a later project stage, this need will be further accommodated when different functions will be allocated to different "actors" i.e. computer tools, humans, teams etc.

Another point to make is that concurrency is shown in figure (3h). Refinement of the term in later project stages may mean refinement of the figure and possibly the notation.
Figure 3g: The Use Scenario 1 process (functional perspective) represented using the SGPM notation

Figure 3h: The Use Scenario 1 process – timeline view
Relationship to Task 3.3

In Task 3.3 the use process model (as defined in T3.2) will be enhanced to allocate the roles of people and tools. Figure (31) gives an example on how the processes in US1 shown in figure (3g) can be modified to accommodate that need. In figure (31), the T3 3 specific notation is used to represent the different agents that act on the use processes.

Figure 31 Identification of the role of humans and tools in the US1 process
3.3 Work Package 6 (WP6): 'Demonstrate with representative mechanisms'

3.3.1 Objective

The purpose of Workpackage 6 exercises are set out in the Technical Annex, which states:

"This Workpackage has the objective to systematically and incrementally demonstrate that the emerging standard and data exchange interfaces, within the context of the described process, actually facilitate concurrent engineering in an effective way" (SEDRES Technical-Annex, p33).

3.3.2 General approach

Figure (3j) below provides an overview of the activities during a single Workpackage 6 "Perform ‘real’ design work" data exchange. It should be noted that the overall flow of design information during WP6 Tasks 6.5 - 6.7 data exchanges is not linear but circular. Design information is transferred from the original (Tool A) model via the SEDRES DEM to Tool B. Following further design work (in Tool B), design information is then transferred back to Tool A, where it is incorporated into the original design.

Figure 3j: Overview of WP6 activities [8]. A ‘one to many’ transfer (one design being exported to many recipients) is represented here. The terminology in the ‘time’ arrow relates to the contents of the observers’ and engineers’ checklist headings.
3.3.3 Data collection by observation (theory and practice)

3.3.3.1 What is evaluated

The evaluation exercises are intended to monitor *Product* and *Process*. In this case, the term 'product' refers to the software products which form the SEDRES Data Exchange Mechanisms (DEMs). Their evaluation covers issues such as interface usability, accuracy and completeness of the data model (number of errors etc.), and the speed and reliability of transfers.

The term 'process' refers to systems engineering process improvements resulting from the use of the SEDRES DEMs [8]. Of particular interest is the extent to which their use facilitates effective concurrent engineering.

Analysis of the evaluation data collected through the methods and techniques explained in this chapter will allow the Data Exchange Mechanism and its effects on systems engineering process to be monitored against previously declared measures of effectiveness.

The observations mentioned in this section, were performed by on-site engineers and the results were sent directly to LUTCHI. In this project task, the author took over work from a previous LUTCHI employee and performed most tasks as part of the current team. The main tasks were: collection and organisation of observation documents, encoding of those documents, data analysis, and presentation of the evaluation results at the project evaluation workshop.

3.3.3.2 Monitoring of Forms

During the Use Scenario Perform 'Real' Design work exercises, engineers were required to fill in the set of forms listed below. The contents of these forms were monitored by LUTCHI and used as evaluation data.

- Engineer's Record and Checklist
- Query / Response / Transfer Problem Notes
- Import / Export Form

Appendix A contains the complete set of data collection forms to be filled in by the engineer.
3.3.3.3 Observation

At several points during the design exercises, the engineer was observed by an observer. These observers passively recorded actions taken by the engineer using the Observation Record Form and Checklist (see Appendix A), and did not interfere in any way with the engineer's tasks. Observations could then be clarified through a follow-up question session.

Engineers were also required to continuously make observations on their own work and record these using the Engineer's Record Form and Checklist seen in Appendix A.

Observation Preparation

The following is a set of activities which should be carried out prior to the observation sessions taking place. This should be seen as a strategy for preparation of the observation work and is presented in the form of a checklist (see figure 3k on the next page).

Observation Follow-up Questions

In cases where an action recorded by the observer requires further explanation, observers were expected to note this fact. Immediately after the session being observed has ended, a follow-up interview would take place between the observer and engineer, during which points noted by the observer could be clarified. These questions could not have precisely been defined in advance, and the observer had to devise them so as to acquire factual information as distinct from subjective views or opinions.
Observation Preparation Checklist

1. Nominate participant and external observers at all sites involved in the Use Scenarior Perform 'Real' Design Exercises

Observers are required to monitor the design activities for each Use Scenario exercise. They are required to capture the data using the standard paper-based forms to be found in the SEM.

2. Observers should be trained to note the relevant data and ask appropriate clarification questions

The training workshop addresses these issues. All observers must be familiar with the contents of this document and the SEDRES Evaluation Manual. External observers will also require instruction in the use of follow-up clarification questions. Contact LUTCII for support and advice whenever appropriate.

3. Acquire Support Materials for Observations

Checklists providing support to the observation exercises are required and are provided in the SEM. These should be studied carefully prior to carrying out the observation exercise. Ensure that sufficient numbers of forms are on hand for each observation session.

4. Identify observation points

Observation sessions should be carried out at the critical points in the exercises, for example, at Import and Export of the DEM. As it is impractical and expensive in manpower to have an external observer present at all times it will be important to agree with the relevant partners which points are to be observed by the external observer. The participant observer-engineer.

5. Ascertain protocol for transfer of observation forms

All completed observation record forms for both participant and external observer should be sent to the Evaluation Co-ordinator at LUTCII.

6. Determine protocol for feedback of observation data

Review points should be scheduled in the overall plan of the exercise when feedback will be provided by LUTCII after preliminary analysis. The quality of the feedback will be dependent upon the quality and completeness of the data gathered.

Figure 3k: Checklist for performing an on-site observation
3.3.3.4 Overview of Data Collection Techniques

Figure (31) [8] shows the data collection forms and design snapshots in the context of the exchange cycle shown in figure (31) (the transfer represented here contains only a single recipient for clarity). This shows the scope of each form (the set of activities each form is used to monitor) and the points at which design snapshots and transfer reports are produced. The purpose of each form is also shown.

Figure 31: Overview of data collection activities within the context of an exchange. The scope of each form is represented as the region of activities between the arrows either side of the forms name. The purpose of the data collected by each form and printout is also represented (see key).
3.3.4 The NUDIST software and its role in the evaluation exercise

The use of codes as a means of categorising and interrogating data has always been a key part of qualitative data analysis methodology [8]. Their use has traditionally involved the establishment of complex paper filing systems containing large numbers of 'record cards' and, in many cases, many copies of relevant parts of the data.

The use of word processors has provided a means of simplifying data storage and duplication, but their usefulness as a tool for coding and retrieval is limited.

To address this problem, specialist qualitative data analysis packages have been developed. QSR NUDIST version 4 imposes no restriction on the analysis method, and allows any number of documents to be imported, coded, and analysed. The software can handle all types of qualitative data, including the structured evaluation data to be collected during SEDRES evaluation exercises.

![Figure 3m: A screen shot from the NUDIST software](image-url)
3.3.5 Evaluation procedure (by example)

As outlined above, in cases where a significant amount of structuring has been imposed on data, and the issues to be addressed are known in advance, a set of codes can be established prior to analysis.

NUD.IST provides support for the implementation of such a coding system. Each code or category is stored as a node within a hierarchical 'tree' structure (for example, the Requirements would be children of the Measures of Effectiveness, which would be children of the SEDRES Sub-missions). Nodes for each of the requirements and each of the checklist items could be set up prior to analysis, and any emerging patterns or themes relating to these could be incorporated into the tree structure as children of the related node(s).

Nodes contain a copy of each piece of data (text) coded at that node (belonging to the category described by that node). This is achieved in NUD.IST simply by selecting the relevant text, and choosing (or creating) an appropriate node.

Coding can also be automated through the use of NUD.IST's text search facilities, which can automatically scan a document for a particular phrase, word(s), or related word(s). The results of such a search (and an amount of text specified by the user surrounding each find to supply contextual information) are then automatically copied to a specified (or new) node. This could be used in SEDRES to search for and code, for example, all references to 'semantic errors', or words related to 'semantic'.

3.3.6 Data analysis and results

3.3.6.1 Justification for Investigation and Outline of Result

During the process of document coding, it became apparent that the preliminary evaluation data contained a large amount of information relating to requirements falling under Capability /1 MoE 1: "Is the SEDRES Capability usable?". The volume of data suggested that potentially interesting findings were available.

The information relating to usability requirements was investigated further and compared across different design tools and groups of exchanges for which data has been collected (Use Scenarios 1&2 preparation and Use Scenario 2 exchanges).

The results of these investigations suggested that between tools and exchanges there were some significant differences in the nature of problems encountered. The findings presented here contain sample node contents and examples of how quantitative information about the coding can be used.
3.3.6.2 Detailed Description of the Techniques used and Preliminary Findings

Investigations started with reviewing and reflecting upon the contents of requirements nodes for Capability /1 MoE 1. This enabled all the data relating to each requirement to be considered together, with the option to examine the context of individual text units. This provided an overview of the details surrounding each requirement (for example, what type of changes were being made to Part21 files during rekeying actions), but it was difficult to assess any differences between groups such as the tool in use and exchange context.

NUDIST’s index search facilities were used to sort data for each requirement according to the exchange context (i.e., whether each text unit related to exchanges carried out during US1-preparation, US2-preparation, or US2). This provided a quantitative measure of the number of text units for each requirement and exchange context. This is shown in the table below:

<table>
<thead>
<tr>
<th>Exchange</th>
<th>Requirement</th>
<th>9, Rekeying</th>
<th>10, Reinterpreting</th>
<th>19, Usable (no training)</th>
<th>27, Ambiguity</th>
<th>4, Search Actions</th>
<th>5, Snag Actions</th>
<th>1, Semantic Content</th>
<th>7, Usability &amp; Speed</th>
<th>3, Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Scenario 2</td>
<td>10</td>
<td>11</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>11</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>US1 Preparation</td>
<td>31</td>
<td>6</td>
<td>45</td>
<td>24</td>
<td>2</td>
<td>48</td>
<td>45</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>US2 Preparation</td>
<td>19</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3b: Number of text units referring to each usability requirement for each exchange group

This table is represented by the graph in the next section, along with similar graphs showing usability requirements for import tools and export tools. These graphs were used to direct further investigations into usability requirements, which focused on areas where large volumes of data were present, and where there was an apparent inconsistency between tools or exchanges.
3.3.6.3 Detailed Description of Findings

The graphs below contain the results of comparisons between exchange group and usability requirements, and tools in use and usability requirements. These show the number of text units for each intersection.

Figure 3n: Number of text units referring to each usability requirement for each exchange group

Figure 3o: Number of text units for each usability requirement for each tool as export tool.
Raw data: Number of text units referenced for each intersection

![Graph showing number of text units for each usability requirement for each tool as import tool.]

The last two charts show the volume of evaluation data for each tool during import and export. The large differences in overall volume for each tool are due to inconsistencies in both the type and volume of data collected. For example, the relatively large volumes of observation data collected at SAAB during import of StateMate files into SAO+ explains the general pattern found in the import tool graph.

The three charts do, however, provide 'suggestions' or avenues for further analysis. Examples of these are provided below, along with the results of further investigations of the relevant text units, and samples of this text.
3.3.7 Future work on the NU*DIST database

This section includes the author's own contribution to the evaluation procedure through analysis of the coding scheme and suggestions on its refinement.

3.3.7.1 Limitations of the current approach

![Diagram showing actual, observable, and observable encodable behaviour]

Figure 3q: Information restrictions posed by methods on observation data

The relationship between DEM requirements and database codes is one to be looked at carefully. As illustrated in the previous paragraph not all high level requirements can be easily mapped on observable engineering behaviour. Conversely, not all types of behaviour can be traced back to the DEM requirements. The methods of observation and data collection inevitably shape and often restrict the information that can be gathered as figure (3q) shows. One of the specific problems that was faced relating to the above issue, is that coding tended to become global – often email documents would be coded under the 'observations' section or Part 21 files would be coded in the 'snags' section.

3.3.7.2 Proposed solutions

A proposed solution to the above limitation involves restructuring the database as follows:

- A 'base data' section (as currently is) that records logistical information on all coded documents.
- One tree for each type of document

The tree structures that follow represent the proposed coding for the snags section
Figure 3: Summary of snag-related nodes

Base data (subset of)
  - Export tool
  - Import tool
  - Interface version
  - Model

Import snags
  - Interface crashes
  - Unusable representation
  - P21 file errors

Export snags
  - Interface crashes

Snag context
  - Error message
  - Problem description
  - Reports to ID
  - Assistant
  - Workaround
  - Corrections
3.4 Chapter summary

In this chapter the author's experience with the SEDRES project is presented. Activities were based around two themes: process modelling and evaluation, each yielding a set of valuable points, as follows:

**Process modelling**
- Identification and addressments of process theory and practice
- Communication across design teams and feedback mechanisms
- Co-operation across location, company and tools.

**Evaluation**
- Breakdowns in engineering work in the form of snags
- Data exchange practice
- Realistic limitations of evaluation work
CHAPTER 4

AN EXPLORATORY STUDY OF THE RELATIONSHIP BETWEEN BREAKDOWNS AND DECISION MAKING IN SOFTWARE DESIGN

Table of contents

4.1 OUTLINE OF STUDY .................................. 100
  4.1.1 AIMS .............................................. 100
  4.1.2 BACKGROUND .................................. 100
  4.1.3 METHODOLOGY ................................. 100
  4.1.4 PARTICIPANTS ................................. 101
  4.1.5 PROCEDURE ..................................... 101
  4.1.6 TASK ............................................. 102
  4.1.7 PROTOCOL ANALYSIS PROCEDURE ......... 103

4.2 OVERVIEW OF THE PROTOCOLS ...................... 105
  4.2.1 SUBJECT #3 (S3) .................. 105
  4.2.2 SUBJECT #4 (S4) .................. 106

4.3 ANALYSIS OF S3’S BEHAVIOUR ..................... 107
  4.3.1 TYPICAL BREAKDOWN EPISODES .............. 108
  4.3.2 TYPICAL DECISION MAKING EPISODES ......... 112

4.4 ANALYSIS OF S4’S BEHAVIOUR ..................... 116
  4.4.1 BREAKDOWN ANALYSIS .......................... 116
    4.4.1.1 OVERVIEW .................................. 116
    4.4.1.2 TYPICAL BREAKDOWN EPISODES .......... 118
  4.4.2 DECISION MAKING ANALYSIS ................. 126
    4.4.2.1 OVERVIEW .................................. 126
    4.4.2.2 CHARACTERISTIC DECISION MAKING EPISODES .... 128

4.5 RELATING BREAKDOWNS TO DECISION MAKING ...... 134
  4.5.1 PROTOCOL COINCIDENCE ..................... 134

4.6 CHAPTER SUMMARY .................................. 136
4.1 Outline of study

4.1.1 Aims

The role of this study with respect to the aims of the thesis is to provide us with a better understanding of the software design process at the level and dimension of individual problem solving. Subsequently, to identify ways to explore it and assist it by using specialised software.

In particular, the objective of this study is to identify empirical foundations of aspects of the design process where a designer of intermediate expertise would need some assistance in order to recover from knowledge-related breakdowns. Some interesting types of behaviour apart from breakdowns are: exploration of the design space, patterns of argumentative reasoning, and methodological dependencies. In that respect, this study could be seen as building up on Guindon's work [36].

4.1.2 Background

Not many empirical studies of argumentative reasoning behaviour have been reported, that from Buckingham Shum [11] being the exception. However, that one was focused on implementation rather than the initial planning phase of design. Sonnentag [86] as well as Herbsleb [42] have indeed studied high-level design from neighbouring perspectives like planning and employment of strategies but their studies did not particularly emphasize argumentative decision making. Another interesting strand of protocol analyses of engineering design is reported in [16] Although it is recognised that different engineering disciplines share similarities in terms of their behavioural patterns, considering the uniqueness of design problems, generalisation of results and drawing conclusions from other engineering studies is seen very cautiously, in the absence of a reliable problem classification schema.

4.1.3 Methodology

Verbal protocol analysis (PA) has been suggested by many authors as a suitable method of studying design behaviour, e.g. [1, 20, 24, 37, 86]. In the piece of work reported here, it proved particularly suitable for exploring design behaviour from a novel angle. Gilmore [32] recommends the use of protocol analysis as a hypothesis generating technique and indeed it has been successfully applied in the same methodological context by Kann [50] in his study of concurrent programming.
4.1.4 Participants

Participants included PhD students with substantial experience in Computer Programming, Systems Analysis and other subjects that ensured their high design ability. In contrast to the current research trend that suggests the return to the place of work to study professional designers, it seems that we are unable to do so as their availability is scarce.

The current subjects would be characterised as «intermediate» in terms of their experience. They have a solid academic background and have worked in industry on tasks that involve software design or programming of some sort for at least a year. Although five verbal protocols were collected and analysed, two of them were chosen for further analysis and presentation, here.

4.1.5 Procedure

The study included the following four phases. Note that all the documents that relate to this study are included in appendix B.

Phase 1: Think-aloud training session

The purpose of that session was to:
(a) show the subjects how to think aloud (TA), and
(b) get subjects going in the «rhythm» of talking while thinking, i.e. to get them started.

In that session participants were asked to pick one of three straightforward and small-scale problems (see appendix B) and solve it by verbalising their thoughts at the same time. The experimenter would intervene where appropriate, to prompt or correct them. The suggestion has been made that the experimenter actually shows the subjects how to think aloud by solving a problem segment himself, first. That is expected to be a more efficient method of illustration.

Phase 2: Main task

The purpose of the main (solving) task was for us to observe breakdown and planning behaviour as well as to assess the resources spent on each design aspect.

In this phase designers were given the N-lift problem (see appendix B) and were asked to solve it. Completion time was not important, so they could take as long as they liked - although the experimenter would stop the session if a maximum period of time had elapsed - and arrange their time at will. During the solving process, they were requested to explain what they were doing in a think-aloud manner. They were also asked to make explicit any questions
or queries they might have during that time. The role of the experimenter was occasionally one of a domain expert. He would converse with the subject when asked to and would try to answer questions.

**Phase 3: Debriefing session**

The purpose of the debriefing session was to clarify salient segments of the process that might not have been clear to the experimenter during phase 2 and to get information about implicit activities that had probably not been verbalised. Some general qualitative information was also gathered in that phase.

During that session subjects were asked to converse with the experimenter in an informal fashion commenting on the experience they had just had. Some questions concerning their behaviour were also made (see appendix B).

**4.1.6 Task**

The N-lift problem is fully described in appendix B. It involves the control of a number of lifts so that they move between several floors of a building in order to satisfy customer requests. It is originally attributed to Simon [83]. Guindon [35] and Sonnentag [86] have used it as well, the former being the main inspiration of this study. The reason the same task was chosen is because (a) it is prone to BDs and (b) I can compare and complement the results to those of the above studies.

Its significance lies on the fact that it is ill-defined (for a full description of the term see paragraph 5.2.2) and therefore can be representative of a family of problems found in industrial practice daily. The ultimate objective is to represent the informality and uncertainty that comes with almost every software requirements specification. In addition, the problem is rather complex and this adds up to its realism.

It has to be noted that it is tried to follow the current research trend in the field of the Psychology of Programming which indicates a shift from easy programming tasks to more complex real-world problems. In fact, current trends suggest a three-dimensional shift, that is towards higher task complexity, higher level of abstraction (i.e. high-level design as opposed to programming) and subject expertise (professionals as opposed to students). It must be said that the only dimension in which we are unable to follow and support fully is the third one. Due to practical problems we use «intermediate» subjects i.e. research students with high competence and some but not major expertise.
4.1.7 Protocol analysis procedure

The procedure that was followed to process the protocols was similar to the one used in the SEDRES evaluation in WP6 (section 3.3.5). Although in both studies the theme was recovery from difficulties, the purpose of the analysis was different, in this case. Whereas in the SEDRES evaluation we were looking for breakdown incidents ('snags') and modelled their context of cause, here I aimed at modelling the correlation between breakdowns and argumentative reasoning, a behavioural correlation which was very important to this research but too detailed to be studied in the context of the SEDRES project. That difference of focus resulted in two differences in approach:

- Protocols were analysed both ways separately, i.e. according to two schemes, breakdowns and argumentation,
- The final product would be a process model rather than statistical data,

In particular, the following stages were followed:

Stage A: transcription. In this stage, the taped sessions were transcribed, word for word, in preparation for analysis. The verbatim transcripts were also divided into segments, each corresponding to the expression of one distinct thought.

Stage B: encoding. Here, each segment in the protocol was translated into some consistent vocabulary defined by the experimenter.

Stage C: grouping. In this stage, the explicit clues contained in the encoded protocol were analysed in order to develop a model of the subjects' cognitive activities.

One of the major drawbacks of verbal protocol analysis as a method, is subjectivity. At the encoding phase, it is left to the discretion of the researcher to allocate the tokens to the scheme consistently. A related problem is the quality of the encoding scheme. In order to communicate the results of the analysis to the research community, one needs a pragmatic and interesting scheme. In fact, if you need to make your results comparable, then you need a scheme that has been successfully used before.

All this becomes immensely difficult if one considers the lack of studies of high-level software design/specification - comparing with the number of studies of programming, for example - which results to limited understanding and description of design processes from a cognitive perspective. I based my encoding on clues in Guindon's study [36] - for breakdowns - and clues found in MacLean's study [61] - for decision making. That way my results can be compared to theirs.
As explained in section 2.10, three types of breakdowns have been identified in the literature, namely difficulties (type I), conceptualisations (type II) and general mental model mismatches (type III). Seeing things from a problem-solving point of view, the first two types are of greater concern to this research. In consequence, when identifying a type III breakdown – which would make itself explicit as a shift of focus, or a task interruption, then that had to be classified as either type I, or type II. At the beginning, that was not an easy process as it required extensive scrutiny over the protocols. However, in the process I learned to spot breakdowns more easily and in most cases it was clear when breakdowns were likely to happen. This resulted to a higher accuracy of encoding.

As mentioned before, devising an encoding vocabulary is not a trivial matter; it requires some notion of the task environment as well as familiarity with the actual protocols to be encoded. In addition, once an encoding language is defined and some substantial portion of a protocol encoded according to it, re-coding the protocol if the scheme is modified is tedious and time-consuming. Even more troublesome is the challenge of reliably coding an entire protocol, particularly a long and complex one, according to the defined vocabulary. It is one thing to code the first few statements in a protocol faithfully according to the scheme but quite another to encode statement 1000 as faithfully as statement one. Problems multiply in the grouping/analysis phase because finding patterns in a large data set requires the simultaneous consideration of many statements as opposed to the sequential consideration of single statements required in the encoding phase. If the data set is particularly large or complex, or the patterns in it intricate or infrequent, which is usually the case in programming and design studies, modelling complex processes may be difficult; verbal patterns representing the processes may be widely spaced in the protocol or may be missing steps (which subjects either skipped or failed to mention).

The lack of a simple way to learn protocol analysis methodology can further complicate matters. The technique is generally acquired through apprenticeship, experimentation and examples. But few researchers have the luxury of a teacher or the time for extensive trial and error. In addition, few publications on protocol analysis exist, and even the most comprehensive text on the subject [23] includes only sketchy instructions on practical protocol analysis techniques. It is probably worth mentioning that the life of this study spanned through the whole of the second Ph.D. year.

A few ways are followed to avoid the above problems. One strategy is to concentrate on one small portion of the total task and examine it in detail. Another approach is to sample only small portions of the total information contained in the protocols, scanning for global characteristics or searching for one particular type of clue to the designer's processes. In addition, most researchers reduce the difficulty of protocol analysis by limiting the number of subjects they study; not many studies in the literature have included more than five subjects.
4.2 Overview of the protocols

4.2.1 Subject #3 (S3)

S3 is a typical example of a designer who follows a methodology by the book. In particular, she is a fan of Structured Analysis/Design [97]. By initially decreasing the problem in order to make it solvable she missed out some of the parameters which she identified later on. However, having followed a systematic decomposition strategy she was able to easily consider these new parameters in the solution and modify it, accordingly.

The first such example is a debugging action discovering that the «service request» operation should be performed before «deciding on service». The second example is a forgotten aspect of the problem and that is floor requests. S3 considers multiple floor requests only when the experimenter reminds her towards the end of the session. Checking the design, she performs the necessary modifications relatively easily.

S3 performs simulations in the solution domain in order to test the design for correctness and completeness. This is because she needs to think in a procedural way even when she was testing data flow diagrams (DFDs).

S3 did not face any particular difficulties in proceeding to a solution. She actually had less idle time than any other designer in this study. She knows the success recipe provided by her favourite methodology quite well and there is no hesitation for her to just follow it and produce a design more quickly that the rest of the subjects. Therefore she only encounters two Guindonian (type I) breakdowns: one had to do with coming back to a postponed partial solution and the other one was about an aspect of the problem that was forgotten. In any case, both of these breakdowns could be related to the limited amount of time designers had available to perform the task.

Nevertheless, there are many situations where she encounters a problem that stems from an ambiguity in the problem statement or an incomplete requirement. She quickly goes through that by making assumptions and quick decisions. She therefore goes through a series of Winogradian (type II) breakdowns. That is she resolves an issue via discovering a new aspect of the problem and elaborating the solution, accordingly.

Thus it looks like it is the case that a designer who is very competent to a certain methodology - or to a number of methodologies - does not face many difficulties in proceeding to a design although they might always be prone to other type I breakdowns that have to do with cognitive limitations of humans in dealing with the large amount of information inherent in complex
problems like the one at hand. Of course, designers' limited expertise is an important factor at this point.

On the contrary, type II breakdowns are very common with such designers, especially ones (breakdowns) that lead to the discovery of a new element in the solution. This type of behaviour is attributed to the fact that all methodologies provide to the designer some predefined «generic» primary generators [36] that guide problem decomposition. However, the actual choice of design elements is arbitrary and left to the designer to choose according to the particulars of the specific problem. That is the point where human decision making comes in and type II breakdowns are at play.

It is also worth noting that in the context of a controlled study when we talk about methodology compliance or deviation the term methodology does not necessarily correspond to the methodology as accurately as that is described in a textbook. In fact, it is highly unlikely that this is the case. Therefore in this study which is actually a semi-controlled one, I talk about people seemingly following a methodology, rather than a methodology being applied to its full.

Individual differences in:
(a) the way a methodology has been taught
(b) people's understanding and interpretation of a methodology
(c) the extent of practical expertise on a certain methodology
(d) the types of domains and problems on which it was applied

make a large difference on what would be included in a person's methodology-based design strategy. In other words, it is about personalised copies of certain methodology fragments.

4.2.2 Subject #4 (S4)

An apparent observation on S4 is that he encounters an obstacle in solving the inter-lift communication subproblem, although early in the process he actually mentioned a potential solution but did not follow it. However, considering this problem central to the solution of the main problem, he spends a large amount of his time trying to sort it out instead of probably, leaving it for a moment and trying to work out another aspect of the problem - another subproblem.

This is a generic strategy the subject seemed to be using a few times: «If there is a subproblem which is difficult to solve after devoting some time e.g. ten minutes on it, then leave it for the moment, work on another aspect of the problem, increase your understanding and come back to it later» However, in that instance, he actually spent probably too much time on a simulation activity which was not offering anything new to him.

Consequently, it may be the case that designers of limited expertise with not particularly thorough knowledge of methodologies do not have particularly useful resource allocation
strategies at hand. Such strategies would enable them to pause a certain unpromising activity and review their situation.

S4 gets dragged down this sub-problem ignoring his good start and failing to view the problem from different angles. At that point, an intervention by the experimenter looks like it clarifies the subject’s misconception of the floor buttons mechanism and resolves one of the issues. Essentially, it gives the subject a break/review point which in the end is proved very helpful.

S4’s favourite strategy seems to be of a black-box type and can be summarised as follows. He considers the current (sub)problem as an autonomous processing system with discrete inputs and outputs. First, he specifies the inputs, then the outputs and then the core functional component.

In the quest of producing a process model, subjects’ protocols were analysed individually according to the encoded categories. In this report I have chosen two of the subjects, whose activity is presented in more detail. This choice is based on the following criteria:
(a) subject’s performance
(b) interesting and rich behaviour
(c) behaviour characteristic of at least one more subject in the group
(d) distinct activity from other chosen subject(s)

For each of the chosen subjects, a number of the most interesting breakdowns is presented in a rich format. I then unify patterns of high similarity to form behavioural process models.

4.3 Analysis of S3’s behaviour

In the next section, a detailed schematic description of S3’s behaviour is given, in the form of finite state machines. The rationale behind this choice is the dynamic yet tacit nature of breakdowns and reasoning as opposed to the «habitual» and explicit character of actions like drawing and reviewing. I thus represent BDs as events or interruptions which alter the state of «laborous» action like elaborating a design object.
4.3.1 Typical breakdown episodes

Figure 4a: S3's breakdown episode BD#1
Planning

Check diagram via simulation

BD/I

Adds missing point to design

start

end

Figure 4b S3's breakdown episode BD#2
Figure 4c: S3's breakdown episode BD#3
Figure 4d. S3's breakdown episode BD#4

Figure 4e. S3's breakdown episode BD#5
4.3.2. Typical decision making episodes

**Retrospective view**

Global planning
(recapping what done so far)

↓

Local planning
(Sets out plan for the next course of action)

↓

Decision making
(argumentative reasoning)

↓

Plan execution
(Identifying entities and relationships)

**Introspective view**

Option
(exploring an alternative)

↓

Criterion and assessment

↓

Decision
(carrying on with original choice)

Figure 4f: S3's decision making episode DM#1
Retrospective view

Decision making
(picks objects and associations)

↓

Global planning
(recapping what done so far)

↓

Decision making
(process-related: shall I refine the diagram now or later?)

↓

Local planning
(planning the next course of action)

↓

Plan execution
(involves decision making – performing a simulation he decides on states and transitions)

Introspective view

Option
(exploring an alternative)

↓

Decision
(carrying on with original choice)

Figure 4g: S3’s decision making episode DM#2
Retrospective view

Decision making
(producing algorithm—deciding on which lift will be assigned to service the latest pending request)

Global planning
(conceptualises problem decomposition and prioritises actions accordingly)

Decision making
(goes back to the algorithm and determines stoppage criteria)

Elaborates algorithm

Introspective view

Assertion

Decision

Assessment and Criterion

Decision making
(strategy decision—chooses subproblem to work on)

Elaborates algorithm

Figure 4h: S3's decision making episode DM#3
Retrospective view

Performing simulation
(checking current design element against use scenario)

Decision making
(determines data structures)

Continues with simulation

Introspective view

Assertion

Figure 4i: S3's decision making episode DM#4
4.4. Analysis of S4's behaviour

4.4.1 Breakdown analysis

4.4.1.1 Overview

S4 went through fourteen BDs in total, which included:
(a) difficulties to proceed - no immediate recovery (postponement of sub-solution)
(b) difficulties to proceed - immediate recovery
(c) idea or knowledge generation

In general terms, the type of BD occurrence can very much be predicted (or is dictated) by the stage of the design process the subject is in. Here is how BD episodes are distributed across design stages:

**Understanding the problem** (2).
- Acknowledges difficult aspect of the problem
- Identifies a specific issue and states difficulty in resolving it.

**Performing initial system decomposition** (1).
- Problem too large to solve as is - need to break it down.

**Producing high-level system specification** (8).
- Difficulty in defining inter-lift communication logic (object interaction)
- Difficulty in representing inter-lift communication logic
- Test case scenario #1 of inter-lift communication to be resolved
- Test case scenario #2 of inter-lift communication to be resolved
- Test case scenario #3 of inter-lift communication to be resolved
- Revisiting test case scenario #1
- Revisiting definition of inter-lift communication logic
- Conceptualisation by checking the requirements.

**Converting abstract specification to detailed pseudocode** (2).
- Specification of inputs to 'scheduler' object.
- Specification of processing in 'scheduler' object.

**Putting things together** (2)
- Data structuring and synchronizing.
- Merging partial solutions.
Reviews solution so far and elaborates - final touches (5).

- Reviews specification w.r.t. new test case scenario.
- Checks pseudocode for completeness.
- Finds trouble making sense of previously written specifications.
- Spots ambiguous specification statement - elaborates in detail.
- The timing problem - elaborates on possibilities for implementation.

It has to be said that by no means does the designer follow the forementioned steps in that strict order. His design process is very often serendipitous [35] with him moving from reviewing the specification to clarifying a corresponding requirement by reading the problem statement, or elaborating initial ideas further at early stages. But in retrospect, he is driven by a tendency to balance his work in a systematic top-down breadth-first route, in order to produce maximum results.
4.4.1.2 Typical breakdown episodes

In this paragraph, a set of characteristic examples of episodes for each design stage is presented.

EPISODE 2: Identifies a specific issue and states difficulty in resolving it
STAGE: Understanding the problem statement
TYPE: VII

TEXTUAL DESCRIPTION. The designer builds up an initial approach to the solution internally. That is, every lift is going to have its own request queue out of button presses within the lift, and requests will be served on a First-Come-First-Served basis. A central block of software (scheduler) has already assigned a new floor request to the appropriate (same direction, closer) lift. However, he realises that there is a case he hasn’t catered for and that is intermediate floor requests. He doesn’t know how to sort these out at this point. He postpones the solution of this sub-problem for later.

Makes initial decision on solution. He’s going to have queues and they’re going to work intelligently. This is probably because of a design schema he acquires out of his own work on intelligent control of robot agents creating intelligent decision trees. In fact, this assertion comes back later in form of a breakdown.

**Introspective view**

- Plan (on solution)
- Simulation (checking whether plan is valid)
- Breakdown (faces difficulty in executing the plan)
- Reading problem statement (postpones solution - plan abandoned)

**Retrospective view**

- Reading problem statement in order to understand the problem
- Comes up with an initial solution goal and a subsequent abstract plan
- Performs two simulations, in order to test his plan.
- The plan is right, but its decomposition is difficult enough to cause a problem
- Goes back to reading the problem statement. He’ll come back to his plan later.

Figure 4j S4’s breakdown episode BD#2
EPISODE 3: Problem too large to solve as is - need to break it down.

LOCATION: [m/(N-J)]

TEXTUAL DESCRIPTION: Getting down to solve the problem after clarifying some requirements, the designer realises that he has to somehow simplify it, as it is complex, and starts solving a more workable bit. In the meantime he reveals that he first wants to have a high-level view of the system before getting into detail. He then states his chosen strategy: he is going to work on one elevator and then expand the solution to N elevators. After this decision he straight away follows a personalised «black-box» strategy.

---

**Introspective view**

- Clarifies requirements
- Breakdown (states risen issue)
- Reading problem statement (obtaining holistic view of the problem)
- States generic strategy
- Recovery decision (issue resolved)
- Associated plan ("functional" (black-box) strategy)

---

**Retrospective view**

- Reading problem statement clarifying system requirements
- Initiates solving plan
- Locking for initial decomposition clue
- Chooses "lift" as decomposition factor
- Follows next level strategy

---

Figure 4k. S4's breakdown episode BD#3.
EPISODE 4: Difficulty in defining inter-lift communication logic
LOCATION: [m/(N-J)]

TEXTUAL DESCRIPTION. It looks like S4 might have sorted out the logic of the current sub-problem internally but putting it down in a certain notation and fitting it to the rest of the solution is not an easy task, considering the dynamic character of the system. It might be the case that when faced with a (sub)problem for which no specialised design schema exists (i.e. no similar problem has been solved before by the individual) there is always an associated representational problem (i.e. the issue of how to represent this new problem in terms of the known notations in a way that fits the rest of the system, there’s a central module receiving input from \((2M - 2) \times N\) floor buttons, N-lift direction and position and the task is to assign the ‘current’ (i.e. latest) floor request to the right lift according to lift information. Having read/interpreted the problem statement properly, he’d realised that there are actually \((2M - 2)\) floor buttons, i.e. 2 buttons on each floor but top and bottom regardless of the number of lifts, as he indeed does later and would have made his life easier.

Lack of strategy. S4 realises that there has to be some form of inter-lift communication. In order to solve that, he needs some representational tools while he’s not sure which one to use, either diagrams or pseudocode. (The problem is obviously too large to hold in one’s head. The decision is to go for loose pseudocode and use diagrams where/if needed. This is a result of the lack of design notational knowledge and application of it - which level do these notations fit in. A scenario-based simulation saves the day. Basically, this is a highly reactive system and people have trouble describing the control flow in a single thread in pseudocode.
This episode relates to episode 4. S4 has identified inter-lift communication as a separate subproblem and to an extent has internally defined the logic behind the solution. However, he has a problem representing it - he has a choice between diagrams and pseudocode. After a few elaborations of this issue via scenario and rethinking of the communication logic, (note that although he's got an idea of how he'll go about solving that problem, he hasn't actually fully solved it and he hasn't of course put anything down - i.e. another case/scenario-based issue is pending), he makes the decision to start with verbal analysis (loose pseudocode) using diagrams where/if needed.

Another viewpoint to see this (Guindon) is lack of application of diagrams or pseudocode on realistic problems etc see report.

He realises that although he concentrates at one lift at a time, he has to communicate with other lifts, as well. Hasn't started to solve that yet. Hem however, relates that problem directly to how to represent the solution - treats it like the key issue.
Introspective view

- Makes assumption (elaboration)
- Generates solution goal/plan (elaboration)
- Breakdown (representational question)
  - Elaboration of breakdown issue (investigating the options)
  - Breakdown (the inter-lift communication issue)
  - Performing simulation

Retrospective view

- Outlining solution internally (following black-box strategy)
- Expresses the representational issue (issue #5) as an argument
- States object interaction issue (issue #4)
- Runs short integrating scenario
- Elaborates issue #4
- Elaborates issue #5
- Runs scenario on issue #4
- Elaboration and resolution of issue #5
- Plan for resolution of issue #4 (starts writing algorithm)

Figure 4: S4’s breakdown episodes BD#4-5
EPISODE 4a:
LOCATION: [m/(N-J)]

TEXTUAL DESCRIPTION. This is a come-back to episode 4 supporting it with a specific scenario. Basically episode 4 reflects the general issue of inter-lift communication. However, to be resolved you need to solve a few sub-cases based on different scenarios-simulations and that's what happens here. Another lift is closer to the floor request and travels in the same direction - looks like the request should be passed on to that lift.

Starts simulation in order to get complete picture. Problem: how to schedule current request, in terms of: (*) my route, (*) other lifts, (*) request scenario, (*) how to implement it. He gets stuck to a problem which results from (*) not considering the requirements carefully, (*) not deciding on his own servicing strategy.

He decides to follow a personalised 'black-box' strategy, whose first step is to consider all lift inputs. He does that but then he finds trouble decomposing the main functional component. In order to make that clearer, he performs a simulation in the problem domain which enables him to internally produce a first version of his decomposed system. Instead of putting it down, he continues with the simulation performing a reasoning outline task in the form of a question to the experimenter, i.e. he expresses the BD in the form of an argument. So he's in a problem of (a) interpreting and inferring requirements and making assumptions and (b) making a solvable choice.

![Diagram](image)

Figure 4m S4's breakdown episode BD#4a

123
EPISODE 4a: (revisited)
LOCATION: [m/(N-J)]

TEXTUAL DESCRIPTION: This is the other 'side' of BD4a. It's BD4a rephrased, revisited and resolved. (All these BDs are based on a misconsumption, not to forget) It's the need to avoid a racing condition.

Looks like many breakdowns have a dual character. One of their sides is towards the conceptual (problem domain) i.e. the issue of producing the right logic. Their other side is an implementation (solution domain) view i.e. the issue of how I can realise that logic best notionally, in order to produce a neat problem, i.e. putting together pieces of logic in the best way to form an efficient (?) system. This second side poses problems to S4 as in his resource management time, he quickly tries to assess the difficulty in implementing certain modules of logic with respect to acceptability etc. in order to plan his process. However, that was not always easy unless he actually got down to it (started implementing). Bearing in mind the dual character of software specifications (they are both requirements and design) then it could be said that S4 finds trouble in producing specs for different aspects of the problem and putting them together. This can be seen twice in the document:
(a) on point (1)
(b) on breakdown episodes (4,5)

Performs simulation
(from the user's perspective)

↓

Breakdown
(Difficulty in specifying functional component)

↓

Performs simulation
(from the system's perspective)

↓

Breakdown resolved

Figure 4n S4's breakdown episode BD#4a revisited
EPISODE 4 (revisited).
LOCATION: [m/(N-J)]

TEXTUAL DESCRIPTION: BD4 argument expressed in terms of queue theones. Decision made and BD resolved. As a result, he simplifies his specification. He makes his system implementable - however, he might now not satisfying some of the requirements anymore.

It has to be said that by resolving this one, he avoids BD4, BD5 as well.

The issue here is whether each lift should have their own queue as well - the central queue is certain.

\[
\begin{align*}
\text{Performs simulation} \\
\text{(request allocation scenario)} \\
\downarrow \\
\text{Breakdown} \\
\text{(Considering all the possible scheduling-servicing scencaria)} \\
\downarrow \\
\text{Breakdown resolved} \\
\text{(decision made towards the simplest alternative)} \\
\downarrow \\
\text{Performs simulation} \\
\text{(confirms validity of decision)}
\end{align*}
\]

Figure 4o: S4's breakdown episode BD#4 revisited
4.4.2 Decision making analysis

4.4.2.1 Overview

Decision types

There seem to be 5 types of decision in S4's protocol. These are elaborated as follows:

(I) Resource allocation (how to spend time and effort, how to manage equipment and documents etc.)

(II) Choice of strategy (methodology, notation, heuristic, etc)

(III) System logic related

(IV) Plan related (choice of objects, functions etc)

(V) Structuring and distribution of data and processing (e.g. choice of queues, which module will handle which structure etc.)

Decisions over design stages

Here is how DM episodes are distributed across design stages:

Understanding the problem (3).
- Plans how to use the experimental resources.
- Makes decision on a simplified approach to the solution.
- Performs scenario-based trade off analysis in order to determine acceptable system behaviour.

Performing initial system decomposition (1).
- Decision on how to simplify the problem.

Producing high-level system specification (5)
- Specifies system functionality.
- Makes decision on notation to be used.
- Deciding on scheduling logic.
- Simulation supported decision on servicing logic.
- Refines system architecture
Converting abstract specification to detailed pseudocode (2).

- Elaborates system architecture.
- Decision on how to distribute requests to lifts.

Putting things together (1)

- Decision on data structures.

Reviews solution so far and elaborates - final touches (3).

- Checks and refines pseudocode (defines system logic at the same time).
- Checks and elaborates pseudocode w.r.t. system requirements.
- Makes system timing decision
4.4.2.2. Characteristic decision making episodes

DM EPISODE #1: Plans how to use the experimental documents

TYPE. I

DESCRIPTION. S4 trades off between the two alternative strategies on how to use the experimental documents. Instead of writing onto scrap paper and after getting it right transferring the whole design, he prefers to use the «Design Solution Sheet» straight away.

**Contextual (retrospective) view**

- Reads problem statement
- Performs resource allocation trade-off
- Reads problem statement abstracting key points
- Sorts and writes key points on design sheet

**Introspective view**

- Question: Do I need scrap paper?
- Decision: No.
- Plan: Will use design sheet
- Alternative decision and associated plan: Use scrap paper and at the end transfer to design sheet
- Justification for decision

Figure 4p: S4’s decision making episode DM#1
DM EPISODE #3: Performs scenario-based trade off analysis of acceptable types of system behaviour.

TYPE: III, V

DESCRIPTION: In this episode S4 perform two simulations of system behaviour based on typical use scenarios in order to answer the same question as in episode #2 (degree of queue intelligence). He thinks that considering queues intelligently is a necessary prerequisite in order for the system to exhibit reasonable behaviour. He thus breaches his previous decision.

Contextual (retrospective) view

Clarifies system requirements (by performing simulations)

\[\downarrow\]

Simulates scenario #1

\[\downarrow\]

Identifies issue and makes decision

\[\downarrow\]

Simulates scenario #2 (to back decision)

\[\downarrow\]

Reads problem statement

Introspective view

Question: Intelligence is needed in the way queues are handled

\[\downarrow\]

Options presented as use scenarios

\[\downarrow\]

Decision: Intelligence adopted

\[\downarrow\]

Assessment with respect to user acceptance criterion

\[\downarrow\]

(Another simulation (new test case) backs the same decision)

\[\downarrow\]

Plan

Figure 4q. S4’s decision making episode DM#3.
DM EPISODE #4. Decision on how to simplify the problem

TYPE: IV

DESCRIPTION: The designer decomposes the system in an object oriented manner. Identifying 'lift' as the most important system entity, he considers one lift out of the whole system. Having designed a system that controls that lift, he's going to expand his design to N lifts. That is his decision and associated plan.

**Contextual (retrospective) view**

- Reads problem statement
- Faces initial decomposition question
- Reflects on strategy
- Picks decomposition cue
- States plan

**Introspective view**

- Question: How should I go about solving this?
- Goal: Have to simplify problem
- Question restated
- (Reads problem statement)
- Strategy decision (plan):
  Going to see it from a high-level viewpoint
- Decision: one elevator to be considered
  Associated plan:
  I can then generalise my solution to the rest N-1 elevators

Figure 4r: S4's decision making episode DM#4
DM EPISODE #6: Representational issue

TYPE: II

DESCRIPTION: S4 performs a trade off analysis in order to decide which level of abstraction to take, each level corresponding to a certain notation. He chooses pseudocode rather than diagrams.

**Contextual (retrospective) view**

- Specifies function outline
  - Specifies inputs
  - States representational question
  - Explains intricacy of situation
  - Elaborates argument
  - Resolves issue

**Introspective view**

- Options: pseudocode Vs diagrams
  - Question: What is the best way to represent inputs?
  - (Returns to inter-lift communication (structural) problem)
  - Question, options restated
  - Elaboration of option #2
  - Decision: pseudocode
  - Associated plan: Diagrams may be used where appropriate

Figure 4s S4's decision making episode DM#6
DM EPISODE #8: Simulation supported decision on servicing logic

TYPE: III

DESCRIPTION. By reviewing his spec he comes up with a typical use scenario where the chosen lift that is going to service the current request should notify the rest of the lifts to avoid unnecessary travelling by them («race condition»). He trades off the implementation considerations of the satisfaction of this inferred requirement, and chooses the easiest option, i.e. not to notify the rest of the lifts. He backs this decision with an argument based on his personal experience with using lifts.

**Contextual (retrospective) view**

- Performs simulation
  - Puts issue
  - Resolves and elaborates 'difficult' option internally
  - Chooses 'easy' option
  - Modifies specification
  - Checks specification

**Introspective view**

- Question (put abstractly)
  - Illustration of option #1 via simulation
  - Illustration of option #2 via same simulation
  - Elaboration of option #1 ('difficult' option)
  - Decision in favour of option #2
  - Justifies decision via usage argument
  - Follows associated plan: modifies specification accordingly

Figure 4t: S4's decision making episode DM#8
DM EPISODE #12: Decision on data structures
TYPE: V
DESCRIPTION: This is a decision S4 has to make in order to solve the subproblem of distribution of tasks to different lifts by the scheduler. By performing a mini simulation of a typical test case scenario, he realises that some of the requests (those from within lifts) should have a higher priority than the others (requests from floors). This leads him to decide to have two queues, one for each type of requests.

**Contextual (retrospective) view**

- Reviews design for completeness
- Puts issue
- Realises prerequisite issue
- Performs simulation
- Resolves prerequisite issue
- Modifies specification resolving initial issue

**Introspective view (syllogism)**

- Issue #1: Distribution of tasks to lifts
- Issue #2: How many queues are needed?
- Investigates the option of having an additional queue holding requests from floors
- Performs simulation of his design in order to test that option
- Assertion: Lift requests must have a higher priority
- Resolution of issue #2: There will be 2 queues for each lift; one for lift requests, and one for floor requests
- Resolution of issue #1 by elaborating the specification according to new data organization

Figure 4u S4's decision making episode DM#12
4.5 Relating breakdowns to decision making

I primarily need to know what types of breakdowns I shall be facing in order to work on how to deal with them. As indicated previously and as indeed MacLean hints [61] it is believed that design rationale could be a useful tool in this quest. The study is expected to convey information on:

- types of breakdowns
- foundations of any type of correlation between breakdown and argumentative reasoning
- indications of how argumentation-based "tools" like QOC could be helpful in breakdown recovery/repair.
- Preliminary evidence of the potential usefulness of QOC: (a) trading-off (b) cognitive load management attributes

4.5.1 Protocol coincidence

The following figure shows an overview of breakdown along decision making episodes over the course of the experimental session. It illustrates:
(a) how these two activities are distributed over the stages of the solving process, and
(b) instances where they coincide.

Note. protocol stages
A: Understanding the system requirements
B: Performing initial system decomposition
C: Producing high-level system specification
D: Converting abstract specification to detailed pseudocode
E: Putting things together
F: Review solution so far and elaborates - final touches

Figure 4v Overview of coincidence of breakdowns to decision making over process stages for S4
There can be identified:

- Occurrences of incidents where the designers clearly find difficulty in continuing and there is an explicit recovery process (usually by performing a simulation)

- Instances where the subjects set a question whose answer they find relatively easily via quick decision making,

- Instances where they face an issue which cannot be resolved readily but needs elaboration of some sort. This can be seen as a breakdown expressed in the form of an argument.

With respect to the relationship between BDs and DM, one can see that they coincide in several cases. On the one hand, critical situations or difficulties can be expressed in terms of arguments (and therefore can possibly be resolved). On the other hand, the most critical or important decisions are the ones that involve a breakdown and not the 'routine' ones and therefore there is a point in working on the management of breakdowns. In fact, I argue that given methodological support design rationale formalisms could be used to convert breakdowns from type I to type II, i.e. facilitate a profitable recovery.

Nevertheless, in this matching mechanism, let me consider:

- the 'non-questionable' decisions and indeed all the types of decision making patterns that are related to breakdown behaviour

- the recovery specifics that a breakdown exhibits and the implication of these to a design rationale-based such mechanism.

Another piece of positive evidence is the fact that most decisions - indeed most types of decisions, as well - are followed by a plan, or an associated goal and an immediate plan. That reflects the positive nature of DM in the design process and clearly indicates a role to the neighbouring breakdown episodes considering that most Guindonian BDs account for plan failures.
4.6 Chapter summary

In this chapter, the behaviour of two software designers is analysed in order to model (a) decision making, (b) breakdowns and (c) their correlation. These designers were chosen out of a pool of five participants in a protocol analysis study for their individual characteristics which made their behavioural processes distinct. The one (S3) was a methodology specialist and the other (S4) was a coding specialist engineer. Their profiles are typical of university graduates and I believe that their analysis provides results with interesting implications for software design education and recruitment. S3 exhibited very stable and predictable behaviour whereas S4 went through several peaks and troughs. S3's decision making process took a form of a Q&A session whereas S4 would rather take time to explore alternatives. S3 seemed to avoid certain issues- 'traps' and their related breakdowns by making simplifying assumptions. S3's process was not too creative, whereas S4 seemed to have the right explorative process although his lack of expertise prevented him from producing an outstanding result.

At this point I feel confident with my knowledge of design practice and ready to proceed to experimental work in order to investigate attributes of design rationale that could make a difference in performance on error-prone tasks like the one studied here.
Part C: Experimental work - Expertise transfer using design rationale

CHAPTER 5

DESIGN RATIONALE TOOLS

Table of contents

5.1 INTRODUCTION .............................................................. 137
5.2 DESIGN RATIONALE ....................................................... 137
  5.2.1 DEFINITION .............................................................. 137
  5.2.2 PARADIGMS AND TOOLS ............................................. 138
    5.2.2.1 IBIS AND rIBIS - QUESTMAP - DRAMA, PHI AND PHIDIAS - JANUS .... 139
    5.2.2.2 QOC AND RELATED TOOLS ......................................... 139
    5.2.2.3 POTTS & BRUNS, DRL AND SIBYL, QAR AND DEBATE BROWSER ........ 145
  5.2.3 DR IN SOFTWARE DESIGN ............................................. 149
5.3 THE LOUIS SYSTEM .......................................................... 150
  5.3.1 FEATURES ............................................................... 150
  5.3.2 DEVELOPMENT PLATFORM ......................................... 150
    5.3.2.2 TCL/TK ............................................................. 150
  5.3.3 ARCHITECTURE ......................................................... 151
  5.3.4 USAGE SCENARIA ...................................................... 152
  5.3.5 USER TRIALS ............................................................. 154
  5.3.6 PLANNING WITH LOUIS ............................................... 155
5.4 DESIGN RATIONALE AS A FACILITATOR OF EXPERTISE TRANSFER .................. 156
  5.4.1 WHAT DOES EXPERTISE CONSTITUTE OF? ................................ 156
  5.4.2 REPRESENTING EXPERTISE FRAGMENTS USING DR ................................ 157
5.5 CHAPTER SUMMARY .......................................................... 158
5.1 Introduction

In this chapter, I investigate the potential of design rationale (DR) as a mean of transfer of expertise among software designers. As DR and related research has grown considerably in the past few years - to the point that it is too large to be contained in a single chapter - I try to focus on issues that relate directly to the above theme.

At the beginning, (5.2) DR and the different approaches to it are briefly introduced. Then follows (5.3) a review of a set of the most prominent DR tools, followed by (5.4) the description of a tool developed by the author, which enables DR capture and analysis. I then discuss some views on how DR could be used as carrier of design expertise. This chapter sets the scene for two experiments that follow in the next chapters which test DR's main attributes in expertise transfer tasks.

5.2 Design rationale

5.2.1 Definition

There is no single definition for design rationale. As a basis, Moran & Carroll's [64] view is adopted, that "Design rationale is the notion that design goes beyond merely accurate descriptions of artefacts, such as specifications, and articulates and represents the reasons and the reasoning processes behind the design and specification of artefacts". More specifically, the collective definition given by Moran and Carroll [64] and seen below, provides an overview of the main perspectives under which DR is currently seen.

design rationale--n.
1. An expression of the relationships between a designed artifact, its purpose, the designer's conceptualization, and the contextual constraints on realizing the purpose.
2. The logical reasons given to justify a designed artifact.
3. A notation for the logical reasons for a designed artifact.
4. A method of designing an artifact whereby the reasons for it are made explicit.
5. Documentation of (a) the reasons for the design of an artifact, (b) the stages or steps of the design process, (c) the history of the design and its context.
6. An explanation of why a designed artifact (or some feature of an artifact) is the way it is.

It is apparent that DR is being used in several ways e.g. as a history mechanism or even as an aid to the construction of designs from scratch. My work comes closest to position (5) although it shares and makes points in most of the other views, too. Nevertheless, the view of
DR in this thesis is not restricted to documentation in the traditional sense, but includes aspects of DR that have direct implications for its use as on-line documentation or more realistically as integral part of other design tools, like CASE for example. In addition, I see several more types of knowledge and expertise being embedded in DR to capture the complete context of design deliberation.

From a more practical perspective, the expression of design reasoning as arguments about issues is referred to as "argumentation-based design rationale" [11]. That is the type of design knowledge of interest in the context of this thesis – therefore, as a convention, argumentation-based design rationale will simply be called as "DR".

5.2.2 Paradigms and tools

A thorough description of different DR paradigms can be found in [11, 64]. A few of them are briefly mentioned here - the most important ones - in order to create a context for discussion. With each paradigm, I mention any method, technique or software that is based on it to enable its application. Rittel was the first to advocate systematic documentation of design rationale as part of design [77] and most of the approaches to DR derive from his work. In particular, Rittel made two important claims: first, that many design problems are "wicked", in contrast to "tame" or "benign" problem which can be modelled computationally, and secondly, that an "argumentative process" was the most effective way to tackle such problems. Rittel's theory of wicked problems was picked up by several people, Simon [83] being one of them. He characterizes design problems as "ill-structured" and contrasts them with "well-structured" in terms of four main characteristics given in Table (5a) below.

<table>
<thead>
<tr>
<th>Well-structured problems</th>
<th>Ill-structured problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Complete and unambiguous specification of problem</td>
<td>(1) Incomplete and ambiguous specification of the problem</td>
</tr>
<tr>
<td>(2) Definite criteria to evaluate the solution and mecbnazmable process for evaluating if the solution is reached</td>
<td>(2) No stopping rule - no definite criteria to evaluate whether a solution is reached</td>
</tr>
<tr>
<td>(3) Any knowledge needed by the problem solver can be represented in one or more problem spaces</td>
<td>(3) Many sources of knowledge (problem spaces) that cannot be determined in advance and need to be integrated</td>
</tr>
<tr>
<td>(4) Enumerable set of operators that can change the initial state into another state and there is at least one problem space in which can be represented initial state, goal state, and all intermediate states considered</td>
<td>(4) No exhaustive, enumerable list of operators to reach a solution and absence of predetermined solution path from initial state to goal state</td>
</tr>
</tbody>
</table>

**Examples**

- Checkers, Tower of Hanoi
- Chess, Theorem proving

**Examples**

- Design (software, architectural)
- Planning, Management
- Document and music composition

| Table 5a: Some contrasting features between well- and ill-structured problems |
On the basis of his theory on wicked problems, Rittel rejected the efforts by the majority of design methodologists to automate design reasoning. The argumentative approach looks to enhance design by improving the reasoning underlying it and is aimed at supporting the reasoning of human designers rather than replacing it with automated reasoning processes [27].

Traditional methods or Issue-based methods? In fact, that's a difficult question to answer. Prescriptive approaches seem to provide good starting points towards the novice-intermediate expertise population as well as towards the initial design stages, whereas they do not seem to be very popular with experienced designers [37, 86]. In later stages and as communication needs increase, issue- or inquiry-based methods seem to be gaining place. From another viewpoint, if one forms a tautology between design construction and traditional methods and one between evaluation and issue-based methods, then that might enable us to see them as complementary. In particular, traditional methods are useful in that they provide philosophical perspectives under which to see the problem, notations which enable designers to realise those perspectives and driving operators to apply in order to progress those notational constructs and produce computer-understandable descriptions of the desired product that solves the problem. These are valuable guides especially to novice designers.

It should be noted that the term 'traditional' is used here to collectively describe all methodological contributions to design, including the traditional ones in the standard Software Engineering sense i.e. the ones that preceded Structured Analysis and Design, Object-Oriented methods, etc..

5.2.2.1 IBIS and rIBIS - QuestMap - DRAMA, PHI and PHIDIAS - JANUS

IBIS [77] is a method for structuring and documenting DR and more or less the father of all current DR methods. The central activity of IBIS is deliberation, that is, considering the 'pro's and 'anti's of alternative answers to questions. The questions deliberated are called issues. Proposed answers - including ones that are mutually exclusive are called answers or positions. Statements of the 'pro's of answers are called arguments. The decision as to which answers to accept and reject is called the resolution of the issue. Figure (5a) graphically demonstrates the core structure of this methodology.
The various issue deliberations are connected by a variety of inter-issue relationships. The original IBIS included "more general than", "similar to", "replaces", "temporal successor of", "logical successor of", and others. Graphs with labelled notes and links representing issues and their relationships were used for visualisation. Such graphs, called *issue maps*, were meant to facilitate navigation through the IBIS *problemspace*. An example of the use of IBIS in practice is shown below.

---

**Figure 5a: IBIS generic structure**

**Figure 5b: An example of the application of IBIS to kitchen design**

IBIS [76] is a hypertext system based on IBIS which allows a distributed set of people to simultaneously browse and edit multiple views of a hypertext network that reflects a design discussion. It is a research tool which combines the merits of two other prototype tools.
developed at MCC, namely GROVE and giBIS. In particular, GROVE is a multi-user hypertext system, whereas giBIS is an application-specific real-time groupware product. Conklin [15] claims that real-time group hypertext is the suitable architecture and data model on which to support an environment for the system design process, as “its open architecture especially fits the informality that is characteristic of the upstream of the design process”.

QuestMap is a collaborative hypermedia system [30], based in giBIS (see figure (5c)). Questions, Ideas, and supporting or objecting Arguments are used to visualise discussions, track unresolved issues, and qualitatively assess the strengths and weaknesses of different positions. Also available are links to relevant documents and embedded maps that can encapsulate resolved problems or contain more detailed analyses as backing for a particular node.

![Figure 5c: A screen shot from QuestMap](image-url)

DRAMA (Design RAtionale MAinagement) [7] is a methodology and associated software tool for recording and managing DR (figure (5d)). It is also based on the IBIS model, which it augments in several ways to make it appropriate for engineering design. In particular, support has been added for: articulating and tracking goals; hierarchical structuring into decision trees;
and the use of quantitative (as well as the standard qualitative) argumentation. Having only recently been released, DRAMA is the first commercial DR system for engineering design.

Figure 5d: The DRAMA tool

Another augmentation to IBIS came from McCall [63]. He suggested that there are two related types of information that are omitted from IBIS but that are required for an issue-based approach to serve design effectively. The first and most basic is dependency relationships between issue resolutions, that is, relationships representing the fact that the answering of issues often depends on how other issues are answered. IBIS has no way of representing such dependencies; instead, it treats issue-resolution processes as if they were separable.

The second type of information omitted is questions that are not deliberated, that is, questions for which 'pro's and 'anti's of alternative answers are not considered. IBIS ignores these in favour of those questions with which debate and controversy are most likely to be associated. Yet non-deliberated questions occur frequently in design and can influence the resolutions of issues, as illustrated in chapter 4. Further evidence of this is Potts's work [73] as well as Shum's [12] - it is also a well established view in the discussions that take place in the CSCA discussion list [17]. Furthermore, many such questions themselves have answers that depend on the resolution of issues.
In an effort to overcome these limitations of IBIS, McCall [63] developed the PHI approach of documenting DR. PHI (Procedural Hierarchy of Issues) like IBIS, is a design method rather than a piece of software. It differs from IBIS in two crucial respects: it uses a broader definition of the concept issue and it uses a new principle for linking issues together (see figure (5e)). McCall claims that it implements the argumentative approach proposed by Rittel more completely and also that it represents the structure of the design process more accurately. PHI too has provided the conceptual basis for a number of issue-based hypermedia systems, including MIKROPLIS, JANUS, ViewPoints, AAA, PHIDIAS, and more recently, REFLACT.

Figure 5e: A quasi-hierarchical structure of subissues as the model behind PHI

An example can be seen below.
5.2.2.2 QOC and related tools

QOC is a formalism used to represent design rationale. It is straightforward and fairly expressive - by using simple notions of argumentation you can represent the reasoning behind systems design. As its name denotes, QOC's basic elements are Questions, Options and Criteria. Questions stand for issues that come up during a design task, Options stand for alternative answers to the question at hand and Criteria are meant to be objectives under which each option is evaluated for fitness. Criteria can be assessed either positively or negatively. Here's an example adopted from MacLean et al. [61]:

Generic QOC vocabulary:

![Figure 5f: Application of PHI to the same problem as in figure (5b).](image)

![Figure 5g: QOC generic syntax and grammar.](image)
According to Buckingham Shum [11], the main difference between QOC and IBIS lies in the IBIS emphasis on capturing more of the design process as it unfolds in meetings (as opposed to retrospectively rationalising the DR as in QOC) and for a single design (as opposed to clarifying the dimensions defining the space in which the design sits in relation to other possible designs).

Chan [13] has built a graphical editor for QOC diagrams, which facilitates the generation of networks that represent the argumentation behind formal specifications. That is part of Johnson's work [49] on producing *literate specifications* to accompany formal language specifications in order to overcome some of the latter's communicative shortcomings as faced in large-scale development projects. Johnson's interest lies on the communication of formal language descriptions of human-machine interfaces across design teams and across project stages. In order to overcome some of the related weaknesses of such representations, e.g. "lack of vernacular descriptions" and "lack of justification", he introduces the notion of literate specifications, a set of DR-based descriptions of the system. This builds upon the literate programming techniques that were first proposed by Knuth and later developed by Thimbleby - both referenced in [49]. Knuth and Thimbleby have developed tools and techniques that explicitly link natural language documentation to the code of computer programs. Through that graphical QOC editor, Chan and Johnson provide a hypertext-based natural-language descriptions of specifications and QOC elements.

User trials have shown that designers still need work on how to construct DR fragments especially as the design space becomes larger, which is the case in industrial workplaces. Future versions of the tool plan version control and consistency checks of the design through Prolog-based descriptions of DR.

Although the topic of formal aspects of software design as such is not in the scope of this thesis, Johnson's contribution in the communicative aspects of specifications is very interesting and shares ground with the aim of this research of communicating semiformal, mainly diagrammatic pieces of software architectural designs accompanied by DR. I will thus pick up on their work later on in this chapter and eventually, in chapter 8.

### 5.2.2.3 Potts & Bruns, DRL and SIBYL, QAR and Debate Browser

Potts & Bruns [72] presented a model for relating entities in a software engineering method to IBIS-based design deliberation. In this model, a design history is made up by the network of intermediate artefacts produced en route to a finished design, artefacts being specifications or design documents which are derived from one another through deliberation nodes (represented as Issues, Alternatives, and Justifications). The particular artefacts depend on
the software method being supported. Based in IBIS, Issues drive Alternatives, which drive Justifications. The key difference from other DR representations, namely the integration with software engineering methods, is achieved through deriving artefacts from alternatives. The syntax of the notation is summarised in figure (5h).

![Diagram](image)

Figure 5h: The Potts & Bruns argumentation model

DRL (Decision Rationale Language) is a formalism developed by Lee [58] as an extension to Potts & Bruns's model, and also combining the merits of IBIS and QOC. The main aim was to provide computational support for semiformal DR using the SIBYL tool [57]. The extension consists of enriching the internal structure of justification by making explicit the goals presupposed by arguments, the relations among arguments, and the first-class nature of these relations. In particular, they stipulate Potts & Bruns distinction between decision rationale and design rationale in the following way: a decision rationale is in Potts & Burns terminology a rationale, i.e. the issue, it positions and the justifications. A design rationale, as Lee defines it is a superset of decision rationale; it represents not only the deliberation process for resolving an issue but also the relationship among the resulting intermediate artefacts of such deliberations. An example of DRL is shown in figure (5i)
An interesting attempt to integrate argumentation into a CASE (Computer-Aided Software Engineering) environment in a seamless manner is the QAR (Question-Answer-Argument) method as well as its implementation, Debate Browser [67]. Although QAR is abstracted from various DR methods, it is closest to the Potts & Bruns method [72], in which the process element of systems development is made explicit. Aimed to simplify the explicit rhetorical structure of DR, the QAR method includes nodes, links and hyperdocuments (see figure (5)).
Figure 5j: The QAR method.

A 'node', may represent any of QAR elements and is an information container, which can contain text or other media, information on the creator of the node, or timestamp of creation. There are several types of 'link's in QAR, depending on the type of nodes they associate. For example, relationships between questions are of three types: generalisation-specialisation, replacement, and parenthood-childhood. A node always belongs to a hyper-document, a collection of discussions, consisting of nodes and links between the nodes. There can be various design rationale hyper-documents, i.e., an organisation mechanism for debates going on different kinds of subjects. One of Debate Browser’s interesting features is support for hypermedia functionality (see figure (5k)), known as Linking Ability which integrates argumentation to design diagrams and vice versa. Also the addition of annotation nodes in diagrams is supported by hypermedia functionality.
5.2.3 DR in software design

Although certain aspects of all of the above paradigms look attractive with software design, there are always trade-offs. The good point about IBIS is that it is more direct in capturing DR in a meeting. The PHI extensions are also valid with software design. QOC is also good in the sense that criteria are very much used in software design as objectives and constraints and are semantically compatible with requirements and system properties. However, most of the reasoning in software design is not of a trade-off nature. DRL on the other hand has an extended element of planning in it using goals, which is very useful with software design, although it is more complicated than the others and that poses a problem when designers are called in to learn to use it.

Oinas-Kukonen [67] claims that DR implementations so far have shown that hypermedia with its node-link data structure suits as a technology for implementing a DR tool. In addition, Jarczyk et al. [48] suggest that the advantages and disadvantages of design rationale systems are the same as with general purpose hypermedia systems.
5.3 The LOUIS system

LOUIS (Logging rationale for User Interface designs) is a system developed by the author. It is based on the QOC method [61] and meant to be used as a platform for experimentation on breakdown management.

5.3.1 Features

As prologued in chapter 2, having interest in the knowledge-intensive aspects of breakdowns, leads my thinking to the concept of a central repository of design information based around argumentative fragments, that will:

- serve multiple stakeholders to the process and product
- make multiple knowledge types available to the above agents in the right representation
- provide a good account of the current state of the process

Current features of LOUIS include:

- Links to the representation of the problem
- Annotation and sketching facilities
- Two-level representation of DR: design space and design rationale

5.3.2 Development platform

LOUIS was developed on a SPARC 5 workstation, running Solaris 2 and using Tcl/Tk 7.4 and the C programming language (gcc).

5.3.2.2 Tcl/Tk

Tcl [69] stands for “tool command language”, and is a library as well as a language. As a textual language it is intended primarily for issuing commands to interactive programs such as text editors, debuggers, illustrators, and shells. It is also programmable, so Tcl users can write command procedures to provide more powerful commands than those in the built-in set.

As a library package, Tcl can be embedded in application programs. The Tcl library consists of a parser for the Tcl language, routines to implement the Tcl built-in commands, and procedures that allow each application to extend Tcl with additional commands specific to that application. The application program generates Tcl commands and passes them to the Tcl
parser for execution. Commands may be generated by reading characters from an input source, or by associating command strings with elements of the application's user interface, such as menu entries, buttons, or keystrokes. When the Tcl library receives commands, it parses them into component fields and executes built-in commands directly. For commands implemented by the application, Tcl calls back to the application to execute the commands. In many cases, commands will invoke recursive invocations of the Tcl interpreter by passing in additional strings to execute (procedures, looping commands, and additional conditional commands all work in this way).

Tk is an extension to Tcl which provides the programmer with an interface to the X11 windowing system. Tk extends the built-in Tcl command set with additional commands for creating user interface widgets. Like Tcl, Tk is implemented as a C library package that can be included in C applications, and it provides a collection of functions that can be invoked from an application to implement new widgets in C.

Wish is a simple windowing shell which permits the user to write Tcl applications in a prototyping environment.

The main argument for choosing Tcl/Tk was portability. The LOUIS software runs on any hardware platform, providing a Tcl-based script interpreter is installed. This makes LOUIS highly accessible and also opens the field for a multi-user version in the future. Another advantage of Tcl/Tk is its flexibility and 'gluing' capability as it can readily be combined with any language, e.g., C/C++, Prolog, Java, Perl, CORBA, even Oracle, and the list goes on. The strength of this merger is that the graphical capabilities of Tk can be retained and at the same time tied with proprietary languages providing functionality for specialised application domains.

In the case of LOUIS, C programs were used to handle file structures, as similar software was available from a previous project. Another obvious deciding factor was the fact that Tcl/Tk is freely available as is its documentation, latest patches and related tools.

5.3.3 Architecture

The LOUIS system is made up of the following components:

- **Control Panel** is the component that controls the main task-based activities like creating, modifying, and saving of DR, and controls all the main other components.

- **Design space mapper** is a canvas Tk widget on which issues can be put and moved around, thus providing a discussion aid at high level of abstraction (context).

The **Issue editor** allows for the modification of issues. It can also be used to affiliate a richer textual description to issues.
Design rationale viewer is an editor for QOC-based DR. Questions, options and criteria are implemented as directly manipulable text fields. The buttons that are associated with them can be used to fetch relevant information from the requirements list (problem editor) or the design product itself. Assessment links have been implemented as clickable lines whose invocation changes their thickness indicating the right weight to the assessment.

The Database manager uses a simple flat file structure to record DR elements and retrieval of fragments can also be based on content rather than only on title. That allows for criteria-based searches.

Problem text editor is a simple text editor with added up facilities for highlighting parts of the requirements statement.

Whiteboard annotator is a tool used by users to draw sketches and thus perform simulations or annotate.

5.3.4 Usage scenario

![Diagram of the design rationale viewer interface]

Figure 5: LOUIS scenario 1 - creation of a new issue and its associated rationale.
Figure 5m: LOUIS use scenario 2 - exploring the design space in two levels issue clustering and design rationale management.

Figure 5n: LOUIS use scenario 3 - DR as a supplement of the design. Four types of design knowledge combined: problem statement, (Problem text editor), design product representation (OODesigner CASE tool), design rationale (LOUIS – DR Viewer), and simulation/annotation (WhiteBoard).
5.3.5 User trials

Following the protocol analysis study (chapter 4), participants tried out the LOUIS system in order to perform certain design tasks related to the N-lift problem. Here is a list of the problems that were then faced.

Argumentation model breakdowns

Some users found the argumentation model too imposing. In fact, as explained in chapter 4, design process did not match QOC or Design Space Analysis structure [61] as Buckingham Shum [9] has also noted. I found their work practice incompatible with the issue-based process suggested by the LOUIS tool. It seemed that unless the model of the process — the way to approach sub-problems — would already have the users' consent, designers were not prepared to follow it.

Cultural breakdowns

In the same concept to the above breakdowns, this set of breakdowns were encountered as part of a trial to encourage designers to record their argumentation as they were designing. That proved a problem confirming previous research on the importance of social and cultural aspects of design communities.

User interface breakdowns

A number of problems with the user interface of LOUIS were encountered and many comments were made. A number of the most important user points is listed here.

- One of the users felt that the approach would only be successful if the argumentative structure would be tailored to the problem at hand. They suggested that they could not easily see how a generic model of argumentation can encompass the characteristics of all set of design or engineering problems.

- Another user insisted on the consistency check she expects such a tool to provide. She mentions that she expects the software to help her correct automatically any syntax errors, or notify on unchanged parts of design (inc DR) components or in the case that a change is made elsewhere that has produced an inconsistency.

- Abstraction of key points On the problem editor, although a highlighting facility is available, users suggested that elicitation of keywords should be enhanced by being able
to perform several analyses onto them and manipulate requirements extracts more flexibly.

- Text fields should be made scrollable as many option descriptions were too large to fit in
- Issue groupings should also be made more advanced supporting a set of graph-based issue management operations like copying, pasting resizing of clusters etc.
- Query facilities on issues in terms of any object that they are related in the design context

5.3.6 Planning with LOUIS

LOUIS was not used in the main experimental trials. Problems of technical nature prevented its further development and application. In particular, it was estimated that further development that would produce a fully functional experimental platform would take minimum of four further months. The work that was needed to do that included the following jobs: further development of the issue map — graphical manipulation of the issues was fairly slow; keystroke level system logging and analysis — for pure experimental purposes; upgrade of the database by linking it to a relational or object-oriented DBMS in order to make DR storing more efficient and maintainable.

Given the considerable amount of time that the protocol analysis had already taken, it was felt that the Ph.D. project was already behind schedule and that concrete experimental results were needed in a tighter timescale. The alternatives were: (a) to abandon LOUIS and perform the main experiment on paper and pencil and (b) to pursue the integration of ready-made and freely available software components with the existing LOUIS components. Exploring the second option, three such components were spotted on the Internet. (i) a Tcl-based graphical editor of geometrical objects that could be used as the basis for the new issue map, (ii) a proprietary recorder and player of user actions and (iii) a relational DBMS with an object-oriented implementation. In short, problems with the system software on the machine prevented me from compiling and linking the library components successfully. The system was not set up or configured properly, and the department's technicians and programmers were unable (or unwilling) to help. As this process of exploration took a month, it was then decided that I had already taken out of the LOUIS tool what I could and that the experiment would be based on paper. Further development of LOUIS can provide a very interesting experimental platform. The prospect of future work on the LOUIS system is explored in section (8.3).
5.4 Design rationale as a facilitator of expertise transfer

The traditional view of design rationale is that it serves design if it helps designers to: (a) to improve their own work, (b) to co-operate with other people involved in the design, (c) to understand existing artefacts, i.e. communicate with past designers. I extend this view a bit further by combining these views in order to produce a good understanding of a past design in order to improve yours.

5.4.1 What does expertise constitute of?

There is unanimous agreement in the literature on that there exist substantial differences in behaviour between expert and novice designers both in the construction and comprehension of design material. In a study of programming activity, Adelson and Soloway [1] report an array of such differences, including
- Simulation and note making are only found when the designer has sufficient domain knowledge
- When the designer does not have sufficient experience with the object being designed he/she will develop the constraints of the design in order to get sufficient specificity for the simulation process
- When, as a result of experience, the designer has an appropriate plan he will use the plan rather than formulating constraints, simulating, and making notes.

Although the tasks of coding and designing invoke different Psychological processes [91] I can in this case cite Guindon’s [35] results on experts working on a design problem - the N-lift problem as the subjects that did worse exhibited behaviour that can be associated to that of the novices in Soloway’s study. Data confirming the same claim on differences in performance in comprehension tasks are also plenty, e.g. see Soloway and Ehrlich [84]. Considering the diversity of behaviour observed in Guindon’s study one is tempted to think that even experts differ in performance among themselves and that the traditional distinction between novices and experts - with all its inherent ambiguities - is perhaps too simplistic as there are several sectors on which two or more individuals may differ with analogical effect on performance.

Further to this direction is Sonnentag’s study [86] which examines factors that affect expert behaviour. She demonstrates that expert performance - planning and use of strategies, in particular - can be predicted by work characteristics. Good [34] is led to similar conclusions in her study of visual programming behaviour.

A process always leaves some of its marks on the product, either in the form of exceptionally good performance features, or in the form of user errors, the metaphor/concept, or whatever. In fact, it may be worth seeing any process which solves the problem at hand in a different way as studying that process may contain information cues that trigger some sort of idea, e.g. an aspect of the problem that has not been thought before.
In fact, problems of this type - expertise transfer - are very often faced in industry. The relevant experience from the SEDRES project is that all too often designers must be trained in new ways of doing things. This is an issue far from what formal education usually provides to a new design graduate. Apart from that, a course does not capture that element - “expertise acquired on the job”, as Curtis [18] puts it. People often say “Read an expert’s C program if you want to learn C”. It could be said that the equivalent of that in the high-level design world is the current research objective.

In a quest for an accurate representation of the design process, one cannot go too far. Let us take the Design Methods for example. Studies of Structured Design, e.g. Vitalari [94] show that the documentation associated with these methods is far from capturing the actual process, in fact, they go as far as not recommending them. Object-oriented ways claim to be natural and indeed have outperformed their structural contestants in doing so - see e.g. [71]. But do they actually provide a good process record? Not too much, as shown below. The closest there is to a process is in fact DR, and that is why it is chosen as a starting point in the pursuit.

5.4.2 Representing expertise fragments using DR

As outlined in chapter 5, a central theme in this research is the use of design rationale to manage the breakdowns that take place as part of software design activities. As shown in chapter 4, BDs directly affect the decision making process and subsequently the course of action, and vice versa. A common piece of ground in all types of breakdown seems to be the knowledge of strategies that correspond to the particular problem class (see figure (2k)). In particular, the lack of readily available such knowledge at the cause of the BD and its subsequent discovery at the recovery stage.

DR has been used in several studies to provide readily available design knowledge. Fischer [25] uses 'seeded' modules of argumentation out of previous designs to facilitate breakdowns and subsequent knowledge discovery, and thereby guide designers. Karsenty [52] found DR to be a useful source of knowledge when designers try to re-use design documents in a real project. Guindon [40] gives a thorough description of knowledge types related with software design. In the next two chapters I will be re using a set of argumentative structures which carry different types of knowledge and are put in different processional contexts. In particular, the following knowledge types are of interest:

- domain
- methodological
- general design
5.5 Chapter summary

In this chapter, I have brought together different pieces of work on design rationale, and assess their suitability for disseminating expert design knowledge. Initially, existing design rationale theories are reviewed, as well as the software tools that accompany them. Not many substantial differences exist among formalisms if one sees them in isolation. However, putting design rationale approaches into practice and onto certain design tasks would reveal several problems, either formalism-specific or formalism-independent, as certain studies have already shown (e.g. [12]). In fact, not much strategic information (e.g. methodological) comes with current formalisms, which still makes the transition to an issue-based design process not a straightforward step. To that end, following Fischer's match of (construction Vs evaluation) to (action Vs reflection) [27] I separate these two processes and concentrate on the latter and look at how a designer can gain expertise by reading someone else's argumentation. In the next two chapter, I put certain properties of design rationale to the test and evaluate their appropriateness to design re-use.
6.1 Introduction

The study presented in this chapter investigates the potential of DR to transfer: (a) expertise, and (b) process-related information on how that expertise has been applied before to novice analysts/designers [45]. By making available the train of arguments and thoughts of a previous (possibly more experienced) designer, as well as any associated heuristics that were used, novice analysts should be able to contribute to the solution. There might also be a possibility for effects of transfer of implicit knowledge or skills.

In particular, a task scenario of DR re-use is put together, and what is of concern is: (a) which type of representation is most suitable to a typical design problem, and (b) what the role of the structure of the argumentation material is in such a context. Can we gather any information out of it about the structure of reasoning that has taken place?

6.2 Hypotheses

6.2.1 DR representation

DR comes in different forms. As shown in chapter 5, most approaches to argumentative DR come in graphical form, as inheritants of the IBIS paradigm. Lee [57] and Buckingham Shum [10] have put forward tabular forms, whereas another few of them, including Fischer [26] use a narrative one. In most current systems, which are predominantly based on hypertext or hypermedia, different representations are combined. Although the majority of these systems provide a high-level graphical map with a zooming facility, and a link to textual or tabular description of user-selected rationale fragments, we know little on how designers of varying expertise assimilate different DR notations in different design contexts or tasks.

As a start, it would be interesting to see the effect of these different notational types to the thinking of a novice analyst trying to make sense of a set of DR fragments. In addition, I am interested in how linguistic properties of formalisms are acquired and how performance fluctuates over them.

The QOC formalism [61] - see also paragraph 5.2.2.1 - is used as a platform on which to test different representational aspects of DR. The choice of QOC is based on the QOC’s appropriateness to user interface design tasks as illustrated in [61] and [59]. The following figure shows an example of a DR fragment from the design of the scroll bar of the Xerox Common Lisp (XCL) environment, adopted from [61], which illustrates all three types of representation.
The Question is: "how wide?" and the Options are narrow and wide. The Criteria are screen compactness and ease of hitting with a mouse. A narrow scroll bar is rejected because ease of hitting with a mouse is more important than screen compactness.

My intuition is that a narrative DR form would be more useful in eliciting semantics out of a DR fragment, whereas a graphical one would probably be more suitable when a combination of multiple (related) fragments is required. No prediction is made on the overall performance.
6.2.2 DR structure

On a relevant note, MacLean et al. [61] suggest the roles of the different DR elements. They mention that «the role of Questions is generative and structural, not evaluative. We must next consider how to evaluate the Options and to rationalise the decisions», «The QOC representation emphasises the systematic development of a space of design Options, structured by Questions.» and also that «The QOC representation brings the objectives for the design, in the form of Criteria, into explicit focus.».

This leaves us with a fairly large space of options on how a DR document based on QOC would be read in order to provide a thorough understanding of the material to the level that one can make judgements and suggest alternative proposals towards the design solution. Also to be mentioned that QOC as an abstract formalism, can be used to capture a set of different styles of systems description or thinking. As an example, I use three different narrative styles which can even correspond to issues faced in different design stages: «what is the advantage of [option X1] over [option X2]?», «what is the most desirable system attribute of [a list of given ones]?», «is objective Z met if I decide to [option Y]?». Note that the first narrative is options-based whereas the second and third ones are criteria-based and we are still left with a non-trivial set of structuring options. Obviously, a narrative argument favours linear thinking and that is certainly not always the case in systems design, as documentation exists in several types of representation. Nevertheless, it is interesting to see how the structure of the material together with the order in which it is presented can make a difference in comprehension and subsequent problem-solving performance. Given that prose structure indicates how decisions and assessments of alternatives were originally expressed, differences in performance would indicate the preferable structural condition under which a novice designer/analyst reasons about an existing design in order to improve it.

Two different types of the narrative QOC form are used in order to investigate that issue. One is leading the reader from options to criteria and the other one is progressing from criteria towards options. It is hypothesised that in a reviewing/evaluating mode, the latter is more suitable and would thus increase subjects performance.
6.3 Method

6.3.1 Materials

Subjects were asked to read design rationale fragments and carry out tasks related to the re-use of those fragments. The re-use task is introduced in figure (6b).

<table>
<thead>
<tr>
<th>DR re-use task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehension</td>
</tr>
<tr>
<td>Syntax</td>
</tr>
<tr>
<td>correct reading</td>
</tr>
</tbody>
</table>

Figure 6b: Overview of an experimental design rationale re-use task

It has to be noted that this is a compromised view of a re-use task, considering the common use of the term. Sub-tasks that relate to the selection of the right material like searching, skimming, scanning, and trying out have been done by the experimenter, as to make its re-use appropriate to the experimental tasks.

In particular, a re-use task is seen here as comprehension followed by problem solving. Comprehension is broken in three parts:

Comprehension of the syntax. That involves the correct reading of the material, i.e., making sure there are no questions on what the elements are and how they associate in terms of the syntactic rules of the DR language that is being used (QOC, in this case). An example of a generic question whose answer would involve 'syntactic comprehension' is: «What is the preferable option for resolving issue X?»

Comprehension of the semantics. That includes interpretation of the current argumentation as well as inferring elements not present but implied in the original author's deliberation. An example of 'understanding the semantics' is expressed as a response to the question. «Have we considered argument Y in our syllogism?»
Comprehension of the context. That includes on the one hand comprehension of the 'internal' context, i.e. how several argumentation fragments connect together, e.g. «How would such a decision affect issue Z?», and on the other hand comprehension of the 'external' context of the design, i.e. «What changes would such a decision bring on diagram D or specification S?».

Problem solving, in turn, constitutes of. Contribution to the discussion which can take the form of modification of DR elements and contribution to the design product. Only comprehension was considered in this experiment — the problem solving part is the subject of experiment #2 (discussed in chapter 7). Comprehension of the «external» context is also dealt with in experiment #2.

The stimulus material was drawn out of the Fast Automatic Teller Machine (FATM) problem, included originally in [61]. The FATM problem as well as the task material can be found in full length in appendix C. A set of eight issues were given, of which five were related - two were siblings implied by a third one, and another one was a child of a fifth one (see appendix C). The eight DR fragments were laid out on two sides of A4 paper, and subjects had to browse through them in order to answer the task questions. The reason that fragments were presented to subjects independently without inter-issue associations been drawn was that part of the task was to identify related issues, as explained below. In previous comprehension experiments, e.g. [68], «familiarity breeds contempt» effects have been reported. In order to avoid such effects, the content of the stimulus material was chosen as to be familiar to all subjects, thereby minimising the chance of content-based individual differences. Therefore, in terms of Adelson & Soloway's classification of expertise [1], we have novices working on an unfamiliar object within a familiar domain.

The task material comprised a set of questions referring to different parts and aspects of the DR material and arranged in 3 tasks. Each task included 7 questions, thus giving a total of 21 questions per trial session. As explained above, re-use in this setting is seen as comprehension of the syntax, semantics, and context of DR fragments. Thus each task included three types of questions. Syntactic questions (3 of them) would give a (or part of a) DR element and request another. Therefore, in those questions, subjects had to browse the given material, spot the right fragment, read it correctly, and put the answer (description - name of the element) down. Semantic questions (2 of them) would either give or imply an (or part of an) element and request either another element or an inference on the part of the subject. Similarly, in those questions, subjects had to spot the right fragment, interpret it correctly (more on what would count for a 'correct' answer given in paragraph 6.3.5 later). Contextual questions (2) would give an element and request
another element from an associated issue. Subjects here had to spot both associated issues, understand them, and resolve the task by taking their link into consideration.

6.3.2 Subjects

Subjects were first-year undergraduate students at Loughborough University doing either Information & Computing, or Computing & Management. Both 'computing' parts of these courses offer the same curriculum in that first semester, which includes a course on Systems Analysis and Design and a course on Programming. Twenty subjects volunteered to take part, two of which were part of a pilot study that preceded the main set of trials. A fair prize in cash was drawn at the end of the experiment, to be won by one of the subjects. The experiment was also presented to them as an educationally beneficial experience. The following table outlines the demographics of the sample.

<table>
<thead>
<tr>
<th>GENDER</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>Female</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>16-20</td>
<td>14</td>
</tr>
<tr>
<td>21-25</td>
<td>3</td>
</tr>
<tr>
<td>26-30</td>
<td>0</td>
</tr>
<tr>
<td>31-35</td>
<td>1</td>
</tr>
<tr>
<td>35+</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COURSE</th>
<th>YEAR</th>
<th>A-LEVELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>I&amp;C</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>C&amp;M</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6a: Demographics of the experimental sample

6.3.3 Design

There were 2 independent variables: representation including three conditions (Narrative, Tabular, Graphical) and structure broken down in two levels (Options-based, Criteria-based) which was implemented under the Narrative condition. Both variables were expanded over three types of task questions, syntax, semantics and context. There were two dependent variables: proportion of correct answers and response time. There were also four qualitative measurements, namely intuitiveness/ease to comprehend, suitability, effectiveness, and overall preference.

The representation variable was implemented within-subjects, with subjects being exposed to all conditions in a single trial session - in particular, an incomplete within-subjects design was employed with all possible orders considered [81] - whereas the structure variable was implemented as between-subjects.
6.3.4 Procedure

Training session
In the training session, participants were given a tutorial on all forms of DR using QOC. That included an introduction to QOC-based DR, followed by a presentation of a few examples drawn out of the design of the XCL scrollbar, followed by a short exercise which introduced subjects to the domain of the main task, i.e. the FATM problem. The tutorial is included in appendix C.

Randomisation procedure
A single two-fold randomisation procedure was used to perform on the one hand counterbalancing (in this case random assignment of subjects to order of conditions) for the representation variable and on the other hand, random assignment of conditions to subjects for the structure variable. For counterbalancing purposes, the sample was divided into six groups of three, each group corresponding to one possible order of conditions for the representation variable. Regardless of their serial number, subjects would randomly pick one of eighteen numbers out of a hat. That number would determine their group, and thereby the order in which conditions would be presented to them. In terms of the structure variable, the sample was divided in two groups, odds and evens, each corresponding to each condition, and thereby the picked number would also assign the right structure condition. The actual assignment of subjects to conditions and orders is shown in appendix C.

Rehearsal
In that session, subjects were taken through a set of similar (to the main one) tasks and explained what it is requested of them. They were then asked to briefly try out for themselves, and the experimenter commented on their answers and made sure the procedure was clear to them. It was also explained to them that interpretation of the original author’s argumentation is what was requested and that only was what qualified as a correct answer - their personal views were optional and welcome as long as they complemented correct answers.
Practical session
In that session, participants were handed the tasks one at a time - the session was videotaped and the camera advanced timer was used to obtain response time measurements.

Debriefing session and questionnaire
On completion of the tasks, subjects had an informal discussion with the experimenter commenting on their experience and at the end they filled in a preference questionnaire. That is also included in appendix C.

6.3.5 Scoring conventions

Correct answers were considered only ones that reflect the intent of the original author of the DR. In quite a few cases, subjects would express their own view on the design, in terms of argumentative elements. In those cases, subjects' responses were considered correct only as long as they were accompanied by the correct answer.

Response time includes the time subjects spent looking at the questions as well as the time they spent looking at the stimulus material up to the point they started putting down the answer. Times are in seconds, truncated of any subdivisions. Time subjects spent reviewing a document that did not end to active hand movement, was ignored. Reviewing time that resulted to writing or modification of an answer was added to the initial response time regardless of when it happened. In cases where a review happened just after they had written something and then added something up that did not essentially change their response, then that time was ignored.
6.4 Results

6.4.1 Training session

There were two prime observations in this session:

Some people found it hard to come up with a set of two criteria for justifying having either a single or multiple slots on an ATM (cash machine). It was clearer to them when prompted to see criteria as desirable system attributes, or design objectives. Less subjects found it hard to make up a set of input options for cash amount selection. These observations point us to semantics of QOC elements that are intricate as a first-time subject and should be elaborated extensively in DR teaching.

Comments from some of the subjects on the graphical QOC form suggest that they make assumptions on how a fragment should be read. In particular,

- assessments and option links were interpreted as action sequences, although no arrowed lines are included in QOC. That could be attributed to those subjects' familiarity with arrowed diagrams where sequence is important, like state transition diagrams; thus expecting DR to be read in the same way. A DR reflects free discussion and does not assume any order. In any case, this point raises an important issue in DR research, i.e. temporality, which will be discussed later on in this chapter.

- the position of elements was expected to directly reflect verbal reasoning. In a relevant vein to the previous point, some subjects assumed that the topology of DR elements was semantically significant, thus expecting «heavier» criteria to sit always on top, thereby giving a symmetry to the assessment links. This is an interesting remark as it imposes or rather pre-supposes some formal rules to the language, in contrast to its persistent semiformality advocated by DR researchers as being more appropriate.
6.4.2 Practical session

The SPSS package was used at all stages of statistical analyses together with Kinnear's book [53]. The following table contains a list of the abbreviations that are used throughout this chapter to refer to the experimental conditions.

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAR, N</td>
<td>Narrative (representation)</td>
</tr>
<tr>
<td>TAB, T</td>
<td>Tabular (representation)</td>
</tr>
<tr>
<td>GRA, G</td>
<td>Graphical (representation)</td>
</tr>
<tr>
<td>SYN</td>
<td>Syntax (task type)</td>
</tr>
<tr>
<td>SEM</td>
<td>Semantics (task type)</td>
</tr>
<tr>
<td>CON</td>
<td>Context (task type)</td>
</tr>
<tr>
<td>O-C</td>
<td>Options-to-Criteria (structure)</td>
</tr>
<tr>
<td>C-O</td>
<td>Criteria-to-Options (structure)</td>
</tr>
</tbody>
</table>

Table 6b: List of abbreviations

6.4.2.1 Representation

Correct answers

Description of data

Table 6c includes a summary of the major descriptive statistics for the overall percentage of correct answers, and table 6d includes the same statistics expanded over the three types of tasks.

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAR</td>
<td>59.259</td>
<td>23.025</td>
<td>.000</td>
<td>83.333</td>
</tr>
<tr>
<td>TAB</td>
<td>62.963</td>
<td>19.433</td>
<td>16.667</td>
<td>100.000</td>
</tr>
<tr>
<td>GRA</td>
<td>71.296</td>
<td>26.075</td>
<td>16.667</td>
<td>100.000</td>
</tr>
</tbody>
</table>

Table 6c. Main descriptive measures of correct answers on representation
In every case graphs outperform the other representations, with narratives coming constantly third and tables fluctuating in between with an exception of the context condition where both narrative and tabular representations do equally badly.

**Exploration of data**

In order to produce the right inferential tests it has to be checked that the data conform to a set of conditions. In particular, to perform parametric statistical tests the distribution of scores must conform to the normal distribution. In the figure below, the distributions for the overall narrative, tabular and graphical conditions are examined in order to compare their means. The Kolmogorov-Smirnov normality test (table 6e) poses no problems for narrative and graphical, whereas tabular marginally fails the test. Observing the distribution, though, there appear no extreme values and tendency and dispersion measures seem very close to the other two distributions. Considering in addition that there are more than two samples, gives me reason to be more relaxed about the strictness of that requirement [53] and go ahead with parametric statistical testing.
Figure 6c: Boxplots of the distributions of correct answers over the 3 types of representation

Test of Normality

<table>
<thead>
<tr>
<th></th>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statistic df Sig.</td>
<td>Statistic df Sig.</td>
</tr>
<tr>
<td>NAR .237</td>
<td>18 .009 .871</td>
<td>18 .018</td>
</tr>
<tr>
<td>TAB .197</td>
<td>18 .064 .908</td>
<td>18 .084</td>
</tr>
<tr>
<td>GRA 289</td>
<td>18 .000 .824</td>
<td>18 .010</td>
</tr>
</tbody>
</table>

Table 6e. Testing the normality of the distribution of the percentage of correct answers over DR representation conditions.

The rest of the triples of conditions that correspond to the experimental hypotheses, i.e. narrative Vs tabular Vs graphical representations over task types (syntax, semantics and context) also produce significant results in the normality tests (see table 6f). Therefore, parametric tests are going to be performed on those conditions, as well.
**Tests of Normality**

<table>
<thead>
<tr>
<th>Source</th>
<th>Statistic</th>
<th>df</th>
<th>Sig.</th>
<th>Statistic</th>
<th>df</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAR.SYN</td>
<td>.276</td>
<td>18</td>
<td>.001</td>
<td>.788</td>
<td>18</td>
<td>.010</td>
</tr>
<tr>
<td>TAB.SYN</td>
<td>.363</td>
<td>18</td>
<td>.000</td>
<td>639</td>
<td>18</td>
<td>.010</td>
</tr>
<tr>
<td>GRA.SYN</td>
<td>434</td>
<td>18</td>
<td>.000</td>
<td>608</td>
<td>18</td>
<td>.010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Statistic</th>
<th>df</th>
<th>Sig.</th>
<th>Statistic</th>
<th>df</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAR.SEM</td>
<td>.309</td>
<td>18</td>
<td>.000</td>
<td>.761</td>
<td>18</td>
<td>.010</td>
</tr>
<tr>
<td>TAB SEM</td>
<td>.279</td>
<td>18</td>
<td>.001</td>
<td>.780</td>
<td>18</td>
<td>.010</td>
</tr>
<tr>
<td>GRA.SEM</td>
<td>.372</td>
<td>18</td>
<td>.000</td>
<td>699</td>
<td>18</td>
<td>.010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Statistic</th>
<th>df</th>
<th>Sig.</th>
<th>Statistic</th>
<th>df</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAR.CON</td>
<td>.222</td>
<td>18</td>
<td>.019</td>
<td>.819</td>
<td>18</td>
<td>.010</td>
</tr>
<tr>
<td>TAB.CON</td>
<td>.278</td>
<td>18</td>
<td>.001</td>
<td>.808</td>
<td>18</td>
<td>.010</td>
</tr>
<tr>
<td>GRA.CON</td>
<td>.301</td>
<td>18</td>
<td>.000</td>
<td>.785</td>
<td>18</td>
<td>.010</td>
</tr>
</tbody>
</table>

Table 6f. Testing the normality of the distribution of the percentage of correct answers over DR representation conditions for different types of tasks.

**Confirmatory analysis**

In order to be able to perform the repeated measures univariate analysis of variance (ANOVA), the distribution has to meet certain additional requirements. The most important of these is that the correlations among the scores at the various levels of the within subjects factor are homogeneous. This requirement is known as the assumption of homogeneity of covariance (or sphericity) [53]. If this assumption is violated, the true type I error rate (i.e., the probability of rejecting Ho when it is true) may be greatly inflated. Since Mauchly's sphericity test on the overall narrative, tabular and graphical conditions is not significant ($W(9, 2)=961$, at $p<.730$) I can accept the result of the ANOVA table 6e which leads me to believe that there are no significant overall effects of DR representation to correct answers.

**Tests of Within-Subjects Effects**

<table>
<thead>
<tr>
<th>Source</th>
<th>Type III Sum of squares</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td>Sphericity Assumed 1368 313</td>
<td>684 156</td>
<td>1.598</td>
<td>.217</td>
</tr>
<tr>
<td></td>
<td>Greenhouse-Geisser 1368 313</td>
<td>710 547</td>
<td>1.598</td>
<td>.218</td>
</tr>
<tr>
<td></td>
<td>Huynh-Feldt 1368 313</td>
<td>684 156</td>
<td>1.598</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>Lower-bound 1368 313</td>
<td>1368 313</td>
<td>1.598</td>
<td>223</td>
</tr>
</tbody>
</table>

Table 6g: Results of ANOVA test for overall effects of notation to correct answers

The same is true for all the other partial hypotheses, too. No significant effect of DR representation on syntax, semantics or context tasks was identified. The corresponding distributions co-varied homogeneously, as $[W(9, 2)=813$, at $p<.191]$ for syntax, $[W(9, 2)=.998$, at $p<.985]$ for semantics and $[W(9, 2)=.968$, at $p<.774]$ for context. However, no statistical significance was detected in the subsequent ANOVA tests, as $[F(9, 2)=1.457$, at $p<.247]$ for syntax, $[F(9, 2)=258$, at $p<.774]$ for semantics and $[F(9, 2)=.602$, at $p<.554]$ for context.
Response time

Description of data

Table (6h) includes a summary of the major descriptive statistics for the most sensitive independent variable, response time, over complete tasks (overall performance).

<table>
<thead>
<tr>
<th></th>
<th>NAR</th>
<th>TAB</th>
<th>GRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>49.2139</td>
<td>37.5680</td>
<td>39.8243</td>
</tr>
<tr>
<td>Std dev</td>
<td>8.9584</td>
<td>9.6907</td>
<td>10.0787</td>
</tr>
<tr>
<td>Minimum</td>
<td>33.00</td>
<td>22.83</td>
<td>27.44</td>
</tr>
<tr>
<td>Maximum</td>
<td>60.17</td>
<td>58.17</td>
<td>66.67</td>
</tr>
</tbody>
</table>

Table 6h. Main descriptive measures of response time on overall representation

An overall superiority of tables and graphs over narratives can be observed. Let us then go on to see whether that superiority holds or how it fluctuates over the partial conditions, i.e. over certain types of tasks. Table (6i) includes a summary of the major descriptive statistics for response time, over task type.

<table>
<thead>
<tr>
<th></th>
<th>NAR.SYN</th>
<th>TAB.SYN</th>
<th>GRA.SYN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>55.4890</td>
<td>34.4118</td>
<td>37.7941</td>
</tr>
<tr>
<td>Std dev</td>
<td>24.1966</td>
<td>17.0882</td>
<td>18.6319</td>
</tr>
<tr>
<td>Minimum</td>
<td>22.00</td>
<td>12.50</td>
<td>16.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>97.00</td>
<td>74.00</td>
<td>79.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NAR.SEM</th>
<th>TAB.SEM</th>
<th>GRA.SEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>47.4118</td>
<td>35.6985</td>
<td>46.2396</td>
</tr>
<tr>
<td>Std dev</td>
<td>12.4384</td>
<td>11.6157</td>
<td>11.5765</td>
</tr>
<tr>
<td>Minimum</td>
<td>28.50</td>
<td>18.00</td>
<td>28.50</td>
</tr>
<tr>
<td>Maximum</td>
<td>72.50</td>
<td>62.50</td>
<td>68.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>NAR.CON</th>
<th>TAB.CON</th>
<th>GRA.CON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>44.7411</td>
<td>42.5938</td>
<td>35.4393</td>
</tr>
<tr>
<td>Std dev</td>
<td>12.1296</td>
<td>15.6808</td>
<td>16.1869</td>
</tr>
<tr>
<td>Minimum</td>
<td>27.50</td>
<td>12.50</td>
<td>15.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>66.00</td>
<td>70.50</td>
<td>74.50</td>
</tr>
</tbody>
</table>

Table 6i Main descriptive measures of response time on representation over task type

In every case tables outperform the other representations, with narratives coming constantly third and graphs fluctuating in between with an exception of the context condition where graphs do much better while both narrative and tabular representations do equally worse.
Exploration of data

In terms of the overall results, i.e. representation over tasks, we come across a similar situation with that in 'correct answers', i.e. the distribution of tabular scores is not apt to normality, as shown in the following table.

<table>
<thead>
<tr>
<th>Tests of Normality</th>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>df</td>
<td>Sig.</td>
</tr>
<tr>
<td>NAR .165</td>
<td>18</td>
<td>.200</td>
</tr>
<tr>
<td>TAB .101</td>
<td>18</td>
<td>.200</td>
</tr>
<tr>
<td>GRA .195</td>
<td>18</td>
<td>.070</td>
</tr>
</tbody>
</table>

Table 6j: Testing the normality of the distribution of response time over DR representation

We thus observe its correlation to the expected normal distribution on the following scatterplot. On the same rationale as above, and considering the high number of groups, it is decided to perform parametric statistical testing.

Normal Q-Q Plot of Tabular

![Normal Q-Q Plot of Tabular](image-url)

Figure 6d: Scatterplot of the correlation between two distributions of response time: observed Vs normalised.

As far as the three triplets of variables that correspond to the secondary hypotheses are concerned, i.e. effect of representation on response time over task type (syntax, semantics and context), similar results are obtained on normality tests and thus it is decided to go ahead with parametric statistical testing.
Confirmatory analysis

A. Overall performance

A univariate within subjects ANOVA yields the following results:

Mauchly's Test of Sphericity

<table>
<thead>
<tr>
<th>Source</th>
<th>Mauchly's W</th>
<th>Approx. df</th>
<th>Sig.</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td>.811</td>
<td>3.351</td>
<td>.187</td>
<td>.8496</td>
</tr>
</tbody>
</table>

Tests of Within-Subjects Effects

<table>
<thead>
<tr>
<th>Source</th>
<th>Type III Sum of Squares</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphericity Assumed</td>
<td>1373.297</td>
<td>686.649</td>
<td>.8496</td>
<td>.001</td>
</tr>
<tr>
<td>Greenhouse-Geisser</td>
<td>1373.297</td>
<td>816.397</td>
<td>.8496</td>
<td>.002</td>
</tr>
<tr>
<td>Huynh-Feldt</td>
<td>1373.297</td>
<td>743.883</td>
<td>.8496</td>
<td>.001</td>
</tr>
<tr>
<td>Lower-bound</td>
<td>1373.297</td>
<td>1373.297</td>
<td>.8496</td>
<td>.010</td>
</tr>
</tbody>
</table>

Tables 6k: Sphericity and subsequent ANOVA tests for response time (overall).

I thus discover significant performance differences among groups exposed in different representations. I then perform post hoc pair-samples t-tests in order to spot specifically which pairs of scores differ significantly. The results of those tests are summarized in the following table

Paired Samples Test

<table>
<thead>
<tr>
<th>Paired Differences</th>
<th>Mean</th>
<th>Std Deviation</th>
<th>Std. Error Mean</th>
<th>df</th>
<th>Sig. (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1 NAR - TAB</td>
<td>11.6459</td>
<td>10.4897</td>
<td>2.4724</td>
<td>4.710</td>
<td>17</td>
</tr>
<tr>
<td>Pair 2 NAR - GRA</td>
<td>9.3896</td>
<td>12.0889</td>
<td>2.8494</td>
<td>3.295</td>
<td>17</td>
</tr>
<tr>
<td>Pair 3 TAB - GRA</td>
<td>-2.2663</td>
<td>15.1240</td>
<td>3.5648</td>
<td>-633</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 6l: Paired samples t-tests for response time (overall).

Note that in the Bonferroni method, the «per-family» error rate is divided by the number of planned pairwise comparisons, c (in this case, c=3) in order to protect against inflation of the «per-family» type I error rate. This means that in this case the test statistic has to show significance beyond the 0.05/3 = 0.02 level for a comparison to be deemed significant. Observing table (6l) it can safely be claimed that: (a) the tabular and graphical representations significantly enhanced DR understanding over the narrative one, (b) although
the tabular form was the quickest to comprehend, its difference over the graphical form was not significant.

B. Performance on syntax tasks

Table (6m) shows the results of a sphericity test and ANOVA for the distribution of response time(s) over syntax tasks in terms of representation.

Mauchly's Test of Sphericity

<table>
<thead>
<tr>
<th>Source</th>
<th>Within Subjects Effect</th>
<th>Mauchly's W Approx. Square</th>
<th>Greenhouse-Geisser</th>
<th>Huynh-Feldt Lower-bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td>895</td>
<td>1.778</td>
<td>2</td>
<td>.411 .905</td>
</tr>
</tbody>
</table>

Tests of Within-Subjects Effects

<table>
<thead>
<tr>
<th>Source</th>
<th>Type III Sum of Squares</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig. (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td>4612.780</td>
<td>2</td>
<td>2306.390</td>
<td>.009</td>
</tr>
<tr>
<td>Greenhouse-Geisser</td>
<td>4612.780</td>
<td>1.810</td>
<td>2548.923</td>
<td>.011</td>
</tr>
<tr>
<td>Huynh-Feldt Lower-bound</td>
<td>4612.780</td>
<td>2.000</td>
<td>2306.390</td>
<td>.009</td>
</tr>
<tr>
<td>Lower-bound</td>
<td>4612.780</td>
<td>1.000</td>
<td>4612.780</td>
<td>.031</td>
</tr>
</tbody>
</table>

a Computed using alpha = .05

Tables 6m: Sphericity and subsequent ANOVA tests for response time over syntax tasks.

Again there are significant differences in performance among representational groups. I continue with post hoc pair-samples t-tests in order to spot significant differences between pairs of scores. The results of those tests are summarized in the following table.

Paired Samples Test

<table>
<thead>
<tr>
<th>Paired Differences</th>
<th>Mean</th>
<th>Std. Deviation Mean</th>
<th>Error95% Std. Mean</th>
<th>t</th>
<th>df</th>
<th>Sig. (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.SYN</td>
<td>-21.0772</td>
<td>27.4270</td>
<td>6.4646</td>
<td>7.4381</td>
<td>34.7163</td>
<td>3.260 17 .005</td>
</tr>
<tr>
<td>T.SYN</td>
<td>-17.6949</td>
<td>33.2375</td>
<td>7.8342</td>
<td>1.1662</td>
<td>34.2235</td>
<td>2.259 17 .037</td>
</tr>
<tr>
<td>N.SYN</td>
<td>-3.3824</td>
<td>25.6535</td>
<td>6.0466</td>
<td>-16 1396</td>
<td>9.3748</td>
<td>-0.559 17 .583</td>
</tr>
<tr>
<td>G.SYN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T.SYN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G SYN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6n: Paired samples t-tests for response time over syntax tasks.

Paired samples tests (table (6n)) show a significant superiority of the tabular and graphical formats over the narrative one.
C. Performance on semantics tasks

Table 6o shows the results of a sphericity test and ANOVA for the distribution of response time(s) over semantics tasks in terms of DR representation.

Mauchly’s Test of Sphericity

<table>
<thead>
<tr>
<th>Mauchly’s W Approx.</th>
<th>df</th>
<th>Sig.</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Within Subjects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effect</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOTATION</td>
<td>994</td>
<td>090</td>
<td>2</td>
</tr>
</tbody>
</table>

Tests of Within-Subjects Effects

<table>
<thead>
<tr>
<th>Source</th>
<th>Type III Sum of Squares</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
<th>Observed Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sphericity Assumed</td>
<td>1498.126</td>
<td>749.063</td>
<td>4.505</td>
<td>.018</td>
<td>.731</td>
</tr>
<tr>
<td>Greenhouse-Geisser</td>
<td>1498.126</td>
<td>753.261</td>
<td>4.505</td>
<td>.019</td>
<td>.729</td>
</tr>
<tr>
<td>Huynh-Feldt Lower-bound</td>
<td>1498.126</td>
<td>749.063</td>
<td>4.505</td>
<td>.018</td>
<td>.731</td>
</tr>
</tbody>
</table>

a Computed using alpha = .05

Tables 6o: Sphericity and subsequent ANOVA tests for response time over semantics tasks

Table 6o suggests a significant effect of DR representation on subjects’ response time over semantics tasks. Follow-up paired samples t-tests (table 6p) reveal statistically significant superiority of tables over graphs and narratives.

Paired Samples Test

<table>
<thead>
<tr>
<th>Paired Differences</th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
<th>t</th>
<th>df</th>
<th>Sig. (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAR.SEM-TAB.SEM</td>
<td>11.7132</td>
<td>18.8970</td>
<td>4.4541</td>
<td>2.630</td>
<td>17</td>
<td>.018</td>
</tr>
<tr>
<td>NAR SEM</td>
<td>-1.1722</td>
<td>17.7996</td>
<td>4.1954</td>
<td>.279</td>
<td>17</td>
<td>.783</td>
</tr>
<tr>
<td>GRA.SEM</td>
<td>-10.5411</td>
<td>17.9896</td>
<td>4.2402</td>
<td>-2.486</td>
<td>17</td>
<td>.024</td>
</tr>
</tbody>
</table>

Table 6p: Paired samples t-tests for response time over semantics tasks
D. Performance on context tasks

Table (6q) shows the results of a sphericity test and ANOVA for the distribution of response time(s) over context tasks in terms of DR representation.

Mauchly's Test of Sphericity

<table>
<thead>
<tr>
<th>Source</th>
<th>Chi-square</th>
<th>df</th>
<th>Sig.</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Within Subjects</td>
<td>853.913</td>
<td>2</td>
<td>.119</td>
<td>.429</td>
</tr>
<tr>
<td>NOTATION</td>
<td>.988</td>
<td>.028</td>
<td>.986</td>
<td>.998</td>
</tr>
</tbody>
</table>

Tests of Within-Subjects Effects

<table>
<thead>
<tr>
<th>Source</th>
<th>Type III Sum of Squares</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig.</th>
<th>Observed Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION</td>
<td>853.913</td>
<td>426.957</td>
<td>2.267</td>
<td>.119</td>
<td>.429</td>
</tr>
<tr>
<td>Assumed Geisser</td>
<td>1.996</td>
<td>427.707</td>
<td>2.267</td>
<td>.119</td>
<td>.429</td>
</tr>
<tr>
<td>Greenhouse-Huynh-Feldt</td>
<td>853.913</td>
<td>2.000</td>
<td>426.957</td>
<td>2.267</td>
<td>.119</td>
</tr>
<tr>
<td>Lower-bound</td>
<td>853.913</td>
<td>853.913</td>
<td>2.267</td>
<td>.151</td>
<td>.295</td>
</tr>
</tbody>
</table>

a Computed using alpha = .05

Tables 6q: Sphericity and subsequent ANOVA tests for response time over context tasks

As shown above, in this case although the context-based distributions of representation vary homogeneously, ANOVA tests do not yield significant results.
6.4.2.2 Argumentative structure

Correct answers

Description of data

Table (6r) includes a summary of the major descriptive statistics for correct answers over complete tasks (overall performance). Options-to-criteria seems to be the favourable argumentative structure, although the difference is not large, at first glance.

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Options-Criteria</th>
<th>Criteria-Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>61.1111</td>
<td>59.2593</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>26.3523</td>
<td>22.2222</td>
</tr>
<tr>
<td>Variance</td>
<td>694.4444</td>
<td>493.8272</td>
</tr>
<tr>
<td>Minimum</td>
<td>.00</td>
<td>33.33</td>
</tr>
<tr>
<td>Maximum</td>
<td>83.33</td>
<td>83.33</td>
</tr>
</tbody>
</table>

Table 6r: Main descriptive measures of percentage of correct answers over argumentative structure (overall)

Table (6s) includes a summary of the major descriptive statistics for correct answers, over the three types of tasks. Note that «O-C» stand for Options_to_Criteria and «C-O» stands for Criteria_to_Options. On syntax and semantics tasks, results seem to follow the overall result of superiority of options-based structure whereas criteria-based argumentation produces better results in context tasks.

<table>
<thead>
<tr>
<th>Statistics</th>
<th>SYNTAX</th>
<th>SEMANTICS</th>
<th>CONTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O-C</td>
<td>C-O</td>
<td>O-C</td>
</tr>
<tr>
<td>Mean</td>
<td>66.6667</td>
<td>61.1111</td>
<td>61.1111</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>35.3553</td>
<td>41.6667</td>
<td>43.3013</td>
</tr>
<tr>
<td>Variance</td>
<td>1250.0000</td>
<td>1736.1111</td>
<td>1875.0000</td>
</tr>
<tr>
<td>Minimum</td>
<td>.00</td>
<td>.00</td>
<td>.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Table 6s: Main descriptive measures of percentage of correct answers over argumentative structure over task type
Exploration of data

Tests of Normality

<table>
<thead>
<tr>
<th></th>
<th>Kolmogorov-Smirnov</th>
<th></th>
<th>Shapiro-Wilk</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statistic</td>
<td>df</td>
<td>Sig.</td>
<td>Statistic</td>
</tr>
<tr>
<td>O-C</td>
<td>250</td>
<td>9</td>
<td>.110</td>
<td>.794</td>
</tr>
<tr>
<td>C-O</td>
<td>212</td>
<td>9</td>
<td>.200</td>
<td>.832</td>
</tr>
</tbody>
</table>

Table 6t. Testing the normality of the distribution of correct answers over DR structure conditions (overall).

As can be seen from table (6t), the Criteria-Options distribution just about fails the Shapiro-Wilk normality test. On the same rationale given in section 6 4.2.1, and considering the high power of t-tests, I go ahead with parametric statistical testing. The same is also true with the partial conditions, as table (6u) illustrates.

Tests of Normality

<table>
<thead>
<tr>
<th></th>
<th>Kolmogorov-Smirnov</th>
<th></th>
<th>Shapiro-Wilk</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Statistic</td>
<td>df</td>
<td>Sig.</td>
<td>Statistic</td>
</tr>
<tr>
<td>O-C SYN</td>
<td>.272</td>
<td>9</td>
<td>.054</td>
<td>.807</td>
</tr>
<tr>
<td>O-C SEM</td>
<td>335</td>
<td>9</td>
<td>.004</td>
<td>.752</td>
</tr>
<tr>
<td>C-O SEM</td>
<td>269</td>
<td>9</td>
<td>.059</td>
<td>.812</td>
</tr>
<tr>
<td>O-C,CON</td>
<td>.278</td>
<td>9</td>
<td>.044</td>
<td>.834</td>
</tr>
<tr>
<td>C-O,CON</td>
<td>223</td>
<td>9</td>
<td>.200</td>
<td>.841</td>
</tr>
</tbody>
</table>

Table 6u: Testing the normality of the distribution of correct answers over DR structure conditions for different task types.
Confirmatory analysis

A. Overall performance

An independent samples t-test is performed, the results of which are shown below. No significance in structural differences is produced.

**Independent Samples Test**

<table>
<thead>
<tr>
<th>Levene's Test for Equality of Variances</th>
<th>t-test for Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Sig.</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
</tr>
<tr>
<td>OVERALL</td>
<td>0.16</td>
</tr>
</tbody>
</table>

Table 6v: Results of independent samples t-test for the overall effect of structure on correct answers.

**B. Performance on task types**

**Independent Samples Test**

<table>
<thead>
<tr>
<th>Levene's Test for Equality of Variances</th>
<th>t-test for Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Sig.</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
</tr>
<tr>
<td>SYN</td>
<td>.336</td>
</tr>
<tr>
<td>SEM</td>
<td>.076</td>
</tr>
<tr>
<td>CON</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 6w: Results of independent samples t-test for the effect of structure on correct answers over task type

No significant performance differences are detected after the manipulation of argumentative structure over task type, either.
Response time

Description of data

Table (6x) includes a summary of the major descriptive statistics for response time over complete tasks (overall performance). Options-based structures seem to be quicker to grasp by novices.

<table>
<thead>
<tr>
<th>Statistics</th>
<th>O-C</th>
<th>C-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>48.8704</td>
<td>49.2708</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>11.5063</td>
<td>5.5493</td>
</tr>
<tr>
<td>Variance</td>
<td>132.3943</td>
<td>30.7951</td>
</tr>
<tr>
<td>Minimum</td>
<td>33.00</td>
<td>34.50</td>
</tr>
<tr>
<td>Maximum</td>
<td>64.00</td>
<td>60.17</td>
</tr>
</tbody>
</table>

Table 6x: Main descriptive measures of response time over argumentative structure (overall)

Table (6y) includes a summary of the major descriptive statistics for response time, over the three types of tasks. Results follow the above trends, i.e. superiority of the options-based structure on syntactic and semantic tasks, whereas on context tasks, a criteria-based structure seems to be favouring quicker comprehension of the underlying design rationale.

<table>
<thead>
<tr>
<th>Statistics</th>
<th>SYNTAX</th>
<th>SEMANTICS</th>
<th>CONTEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>53.5000</td>
<td>52.9375</td>
<td>47.1111</td>
</tr>
<tr>
<td>Std Deviation</td>
<td>22.3411</td>
<td>18.5660</td>
<td>11.7041</td>
</tr>
<tr>
<td>Variance</td>
<td>499.1250</td>
<td>344.6957</td>
<td>136.9881</td>
</tr>
<tr>
<td>Minimum</td>
<td>23.00</td>
<td>22.00</td>
<td>30.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>58.00</td>
<td>97.00</td>
<td>67.00</td>
</tr>
</tbody>
</table>

Table 6y: Main descriptive measures of response time over argumentative structure over task type
Exploration of data

Tables (6z) and (6z-a) show the results of normality tests performed on the distributions of response time over structure for the overall and partial experimental conditions, respectively. Although the results are fairly poor, I rely on the power of t-tests to conduct parametrical statistical testing.

Tests of Normality

<table>
<thead>
<tr>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>df</td>
</tr>
<tr>
<td>O-C .222</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 6z: Testing the normality of the distribution of response time over OR structure conditions (overall).

<table>
<thead>
<tr>
<th>Kolmogorov-Smirnov</th>
<th>Shapiro-Wilk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
<td>df</td>
</tr>
<tr>
<td>C-O.SYN .207</td>
<td>9</td>
</tr>
<tr>
<td>O-C.SEM .204</td>
<td>9</td>
</tr>
<tr>
<td>C-O.SEM .159</td>
<td>9</td>
</tr>
<tr>
<td>C-O CON .257</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 6z-a: Testing the normality of the distribution of response time over OR structure conditions for different task types.
Confirmatory analysis

A. Overall performance

Table 6z-b: Results of independent samples t-test for the overall effect of structure on response time

As table (6z-b) shows, no statistically significant effect of structure on response time is detected for overall performance.

B. Performance over task types

Table 6z-c: Results of independent samples t-test for the effect of structure on response time over task type

Similarly, no statistical significance is detected on the effect of structure on response time over syntax, semantics or context tasks.
6.4.3 Questionnaire session

In this session, whose corresponding documents reside in appendix C, subjects' preferences of representation were surveyed by taking measurements for the qualitative variables of intuitiveness, suitability and effectiveness. Table (6z-e) includes the descriptive statistics of this qualitative comparison and figure (6e) includes the associated bar charts of the mean frequencies. In every case, graphs were preferred over tables or narratives. It is obvious that in all variables we get similar values over the experimental conditions. In addition, the distribution of suitability and effectiveness fluctuated in a remarkably similar manner. This leads us to think that subjects did not clearly differentiate between the meaning of these two attributes, or in fact did not see a distinction in the role for these two measures in the experimental setting. Nevertheless, the common result from this survey is taken seriously.

<table>
<thead>
<tr>
<th>Intuitiveness</th>
<th>Narrative</th>
<th>Tabular</th>
<th>Graphical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>2.2872</td>
<td>2.7111</td>
<td>2.9611</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>1.0345</td>
<td>.8996</td>
<td>.7964</td>
</tr>
<tr>
<td>Variance</td>
<td>1.0702</td>
<td>.8093</td>
<td>.6343</td>
</tr>
<tr>
<td>Minimum</td>
<td>00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>3.70</td>
<td>4.00</td>
<td>4.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Suitability</th>
<th>Narrative</th>
<th>Tabular</th>
<th>Graphical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>2.3872</td>
<td>2.6833</td>
<td>2.9611</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>.7746</td>
<td>.7987</td>
<td>.7405</td>
</tr>
<tr>
<td>Variance</td>
<td>.6000</td>
<td>.6379</td>
<td>.5484</td>
</tr>
<tr>
<td>Minimum</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>3.60</td>
<td>4.00</td>
<td>4.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effectiveness</th>
<th>Narrative</th>
<th>Tabular</th>
<th>Graphical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>2.2094</td>
<td>2.7111</td>
<td>3.0500</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>.9006</td>
<td>.8595</td>
<td>.7778</td>
</tr>
<tr>
<td>Variance</td>
<td>.8111</td>
<td>.7387</td>
<td>.050</td>
</tr>
<tr>
<td>Minimum</td>
<td>.40</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Maximum</td>
<td>3.60</td>
<td>4.00</td>
<td>4.00</td>
</tr>
</tbody>
</table>

Table 6z-d. Main descriptive measures of ratings of preference for DR representations
Figure 6e: Bar charts of means of preference ratings for the three type of representation

Overall preference

Figure 6f Pie chart showing proportions of overall preferences of notational form

Overall preference confirms the results obtained from the partial measurements.
6.4.4 Other results

In this section apart from the results that directly correspond to hypothesis testing, a set of results are presented, which are interesting in that they help us understand better DR comprehension and re-use and knowledge transfer.

Problem solving behaviour

Contribution to the solution

It was very interesting to see that within a time period of roughly twenty minutes, first-year students were familiar with the DR material and were able to make judgements and contribute to the design deliberation. The speed of comprehension was also fairly impressive. I can hardly envisage another design approach of any kind being so quick to grasp and so informative of previous design discussion. It is also roughly estimated that at least 40% of the subjects offered their own views, although they were not explicitly asked to so. This shows a high degree of comprehension as well as the strong viewing that was prompted. There is no hesitation in seeing them implementing their ideas on the FATM design. Therefore, I informally observe DR providing a huge boost to the subjects design process and ability. Another interesting phenomenon is the willingness to comment and enthusiasm that reading a «discussion» on a design comparison brought to students.
6.5 Interpretation of the results and discussion

6.5.1 Representation

6.5.1.1 Correct answers

As no significant effect was detected on the percentage of correct answers by any type of representation, the sensitivity of correct answers as a dependent variable is in doubt. First of all, the ambiguous phrasing of question (A3) in task 1—which was in the end not included in the analysis of the data—indicates the importance of how questions are put in this type of tests. Vessey [92] also reports inconclusive results for the same reason in a comprehension experiment. However, there is a consistent dominance of graphs in all conditions in this category and that does imply a tendency of better (correct) understanding of graph-based DR by novices.

Another interesting source of knowledge that could help to the interpretation of these results is Vessey's theory of the 'cognitive fit' [92]. In a nutshell, the 'cognitive fit' argument states that «... complexity in the task environment will be effectively reduced when the problem-solving aids (tools, techniques, and/or problem representations) support the task strategies (methods or processes) required to perform the task». Although this argument is not 'produced' empirically, it is founded on extended review of the relevant literature which makes it very strong.

Vessey's suggestion that task nature determines strategy and appropriate-to-the-problem representations, leads us to think that certain properties of the task questions may have affected subjects' performance under certain experimental conditions. In order to apply the 'cognitive fit' theory to this experiment, I went through all of the task questions and assessed their nature according to Vessey's guidelines. The outcome was that the first two syntax and first one semantics questions were symbolic in nature as they requested a specific DR element as the answer to the question. On the contrary, context questions as well as the last of the syntax and semantics questions were spatial in nature as they required comparisons and assessments of relationships between (qualitative) data. The distribution of the nature of the tasks under that classification is uneven. In summary, though, it could be said that syntax and semantics tasks are on the whole more symbolic in nature and for that would be expected to favour a tabular DR notation, whereas context tasks are spatial in nature thus expected to induce a graph-based strategy and therefore fit better and favour a graphical DR representation.

If we look at the descriptive statistics on table (6b) we can see that this interpretation explains to a satisfactory extent the data in all partial conditions. In particular, the context category is...
the one in which graphs are doing much better than tables — in fact, tables match narratives on this one. In the same vein, in the syntax category, tables are closer to graphs and possess a much more restricted data dispersion. In the end, though, graphs are the winners. The effects of the ‘cognitive fit’ are obviously neutralised in the semantics and overall conditions.

6.5.1.2 Response time

This is an obviously much more sensitive measure. Overall superiority of the tabular form comes as a bit of a surprise in that the majority of DR software tools do not support it (see chapter 5) although its difference from the performance of graphs is not statistically significant. The narrative representation is significantly third

In syntax tasks the same trends as above were yielded. Both these results and the context ones (commented on below) fully confirm the ‘cognitive fit’ theory as on mainly spatial tasks, tables outperform graphs whereas the opposite is true on tasks that are mainly symbolic in nature.

In semantics tasks, tables do significantly better than text or graphs. This leads us to think that a table facilitates quick understanding of the meaning of argumentative structures and thus helps quick decision making. Backed by the subjects’ comments it is suggested that compactness as well as familiarity are the properties of tables that give them that edge.

As said above, in context tasks graphs were the quicker to grasp, although the differences were not significant. The lack of significance in this case is attributed to the fact that context tasks were fairly complex and subjects often found themselves disorientated in the design space looking for the right answer. The positive correlation between the two dependent variables also suggests that people who spent more time to respond were indeed working their way in the design space looking for a correct answer. On top of that, there might be the case that it was not perfectly clear to them as to how to proceed in such a task. Those two factors contributed to fairly high response times comparing to other types of tasks. It might be the case then that such ‘noisy’ factors confounded the effect of the independent variable.

6.5.2 Argumentative structure

No significance was obtained in either test between options-based and criteria-based approaches. This will have to be attributed to the relatively small size of the sample (N=9). In addition, it might be the case that the different approaches were not made evident enough in the phrasing of the DR fragments. In fact, the first part of a fragment would simply state the QOC element, and that would be identical in both conditions. It was the second paragraph of a narrative that would make explicit any assessments of options or design deliberation.
Therefore, it can be hypothesised that by the time subjects read the second part of the fragment they were evenly familiar with the material, thus the edge between conditions would rely on comprehension of a mere 2-3 line paragraph. That would make the associated differences in performance very difficult to expose themselves.

The consistent dominance of the options-based structure to the criteria-based structure is interpreted in two ways.

I start off from Adelson and Soloway's [1] observation of differences between novice and expert designers. One of the differences was that experts tended to check every partial solution to see how it fits with the rest of the sub-solutions, and how well it addresses the initial problem requirements whereas novices tended to step onto the next action as soon as they reach a (partial) solution without evaluating it or checking how good it is. A neighbouring view comes from Gundon's [37] report that 'good' designers have a good overall schematic picture of the design space and where they are in it at each step, whereas inexperienced designers can at times get lost in that space.

Combining these two views it could assumed that novice designers would not often evaluate their solutions or reflect upon their designs and would thus defer from using criteria to evaluate their options in an argumentative space. It would seem more normal for them to keep on an options-based process, given that options semantically correspond more to alternative sub-solutions and are used when ideas or problems (that need ideas to be solved) come across. Criteria on the other hand seem to be more static or persistent entities in the design process and would be akin to requirements or objectives or constraints in traditional software terms.

Another interpretation would be based on the IBIS approach [89] and the thesis that due to mainly cultural background, people in their everyday life tend to go about their problems in an answers- (or trials-) based manner (corresponding to my options-based approach in QOC terms) often neglecting the original purposes or causes of their action – they are too active and not reflective enough. In this experiment, it could be hypothesised that novice analysts in lack of specialised design strategies [37] they resort to their familiar 'simple' strategy of thinking in an options-based fashion.


6.6 Chapter summary

The experiment presented in this chapter tested the appropriateness of three different representations (narrative, tabular and graphical) and two types of argumentative structure (options-based, criteria-based) for design rationale re-use. The task included the comprehension of syntax, semantics and context of the design of a cash machine by novice analysts and demonstrated the following points:

- design rationale due to its simple and generic semantics is in general terms easily comprehended and argued upon by novice analysts

- graphical design rationale is better understood than its tabular counterpart, which is in turn better understood by its narrative counterpart, although none of those differences is statistically significant. Overall, tabular design rationale is quicker to comprehend than its graphical counterpart which is in turn quicker to comprehend than its narrative counterpart, although none of those differences is statistically significant. The same statement is true for syntax tasks, i.e. ones where subjects have to merely read the argumentation fragments correctly. However, in this case tables and graphs significantly outperform narratives. In semantics tasks, i.e. where subjects have to make good sense of the given argumentation and possibly make inferences, tabular notation is significantly quicker to grasp than its graphical or narrative counterparts. In context tasks, i.e. where analysts have to identify relationships between given issues and make collective judgements, graphical rationale is quicker to comprehend than its tabular and narrative counterparts but none of these superiorities is statistically significant.

- an options-based argumentative structure suits novices better than a criteria-based one, although none of the associated differences in performance were statistically significant. Note that an options-based structure leads the reader from options to criteria whereas the reverse is true in a criteria-based structure.

- graphical form is the subjects’ favourite notational option for design rationale, with tabular and narrative following. The same result was obtained in terms of the subjects’ votes on the categories of intuitiveness, suitability and effectiveness.

- Vessey’s ‘cognitive fit’ argument (see paragraph 6.5.1.1) stands true in design rationale research
CHAPTER 7

THE SUITABILITY OF ARGUMENTATIVE STRUCTURE FOR THE COMPREHENSION AND ACTIVE RE-USE OF DESIGN DOCUMENTATION

Table of contents

7.1 INTRODUCTION ................................................................. 194
7.2 GOALS.................................................................................. 194
7.3 APPROACH............................................................................ 194
7.4 METHOD.................................................................................. 195
  7.4.1 MATERIALS........................................................................ 195
  7.4.2 SUBJECTS.......................................................................... 195
  7.4.3 DESIGN............................................................................... 196
  7.4.3.1 TASKS........................................................................... 196
  7.4.4 PROCEDURE....................................................................... 198
  7.4.5 SCORING CONVENTIONS.................................................. 198

7.5 RESULTS.................................................................................. 199
  7.5.1 TRAINING SESSION......................................................... 199
  7.5.2 PRACTICAL SESSION....................................................... 199

7.6 DISCUSSION, RELATED AND FUTURE WORK......................... 200

7.7 CHAPTER SUMMARY.......................................................... 202
7.1 Introduction

Research on design rationale was felt to be a good starting point in the pursuit of argumentation-based rhetorical remedies for design process breakdowns and improvement, in general. As outlined in chapter 5, this is the second in a series of experiments that aim to compare alternative approaches to aspects of DR that can enhance design performance. In particular, in the study described in this chapter, I test a set of hypotheses that are central in pursuit of the attributes that make DR useful in software design re-use, and that involves the structure of context-based documentation. In experiment #1 (chapter 6) I established a framework of essential structural properties of DR comprehension in the design of an interactive system. Next, follows the study of a complete re-use task placing more emphasis on the second part of re-use, i.e. problem solving behaviour, or 'reading for a purpose', in a more complex design task, the N-lift problem. From a more practical perspective, it is researched whether DR is an appropriate mechanism for transferring expertise as a companion to a standard form of documentation.

7.2 Goals

The goals are influenced by the ease with which DR material was assimilated on experiment #1 (chapter 6), how easily subjects made inferences on the material etc. Given Guindon's and my evidence (chapter 4) that those error-prone points are not due to individual characteristics of the designers but rather inherent in the problem (for example, stemming from the way the system is going to be used) then current designers are expected to make use of the knowledge that is included in the DR material and on the one hand get over the problems (regardless of whether they explicitly face them or not) and come up some creative design ideas. I envisage to show that DR induces contextually rich information to the reader even when that information is not explicitly represented.

7.3 Approach

The adopted approach is as follows: I have identified in the protocol analysis study (chapter 4) a set of breakdowns or points for improvement on the solution of the N-lift problem. In terms of design behaviour, I know that these points are (a) error-prone, (b) decision making points. The investigation lies on presenting new problem solvers with expertise from another one's process and see how helpful it is in resolving those points. The aim is at a mechanism that provides a communication of design contextual expertise towards later stages or new people on a project.
The prime intuition is that making the information explicit in terms of argumentative reasoning, i.e., in the form of a 'virtual discussion' - one that might have possibly taken place - makes the information much more assimilable and context-based. It allows for inferences to take place, which bases my second intuition about the properties of DR. Obviously engaging discussion would probably augment the power of the DR material, but in this context I start off with individuals and how they could re-use DR material. In a setting where the stakeholders are two 'generations' of software designers, I am testing the benefits of DR in design progression.

7.4 Method

7.4.1 Materials

As with experiment #1, the language used is QOC. The rationale is the explicitness of criteria, which makes it suitable for the experimental task. All materials used in this experiment can be found in appendix D.

The subjects went through two problems: the design of a bank transaction system and the N-lift problem. The stimulus and task material comprised.

(a) the design documentation which included: (i) an entity-relationship diagram with N-nary associations (or class diagram in OOD/OMT terminology) of the lift control system, a set of state transition diagrams describing the functionality of the major system objects, as well as a module of (fairly abstract) pseudocode specifying the behaviour of one of the above objects, and (ii) a data flow diagram of a bank transaction system. In both cases (i) and (ii) an A4 page including the problem statement for both tasks was handed out.

(b) The design rationale documentation, i.e., a set of four issues and their associated deliberation, as well as a 'context' diagram showing a high-level view of how these issues relate. That material belonged to the argumentation expressed in the protocol analysis study (chapter 4) on the points where those designers faced difficulty in.

7.4.2 Subjects

In this phase, 3 subjects participated. They were research students at Loughborough, one at the Dept. of Computer Science, one from Manufacturing, and one from Electronic and Electrical Engineering. They all have considerable expertise in software development, and limited expertise in high-level software design.
7.4.3 Design

A between-subjects design was employed. Although the availability of subjects was not high, the irreversible nature of the tasks posed by the need for DR training prior to the main session, favoured the choice of this type of design. Independent manipulation was performed through the availability of design rationale documentation: in the control condition subjects would receive the design documentation only, whereas in the experimental condition they would additionally be presented with the DR documentation.

7.4.3.1 Tasks

Subjects were given two problem tasks of different nature. Initially, they were presented with the bank transaction system (BTS) and then with the N-lift system (NLS). The first problem was data flow-driven and the second problem was event-driven. Although at this stage no counterbalancing was performed, in a proper experimental situation the nature of the problem could be manipulated as a within-subjects factor. That would give the opportunity to compare the results more finely between the types of problem. In both problems, the same (sub) tasks were included, as explained below:

- task 1 was a fill-in-the-blanks test. Certain captions and names of states and data flows were missing and subjects were required to fill them in. In all cases, the task required specific domain knowledge. However, half of the missing bits referred to decisions that had been made during the PA study (chapter 4) at points where certain issues were raised, as a result of problems or difficulties, or as a result of completion of a certain action and review of the next action plan. The rest of the points reflected decisions that solely concerned static properties of the system in question: requirements, objectives, features, or constraints. – criteria in DR terms. It is the former type of design elements which is expected to see DR material to be of most help with. That is because no context on those decisions is included in (or can be directly inferred from) the design documentation itself (apart from decisions and objective prioritisation), whereas there is more such information included in the DR documentation (e.g. arguments). Table (7a) includes an analysis of process elements in terms of their existence in documentation and is derived from my experience with the SEDRES project and the PA study (part B of the thesis). Certain DR approaches like DRL which include an even wider range of elements, e.g. goals and plans, were considered inappropriate for this context. Dependent variables included the percentage of correct answers (prime) and the rate of active design work per 3-minute periods (secondary – higher sensitivity).
• In task 2 subjects were asked to expand the design. On the BTS problem, they were asked to expand one of the 1st level processes («cashier» process) to a 2nd level, and on the NLS problem they were required to write a piece of pseudocode that specifies the functionality of one of the objects («scheduler» object). On this task, the experimenter would rate subjects' solutions in terms of their completeness and their degree of satisfaction of the system requirements. The given mark would provide a dependent experimental measurement.

• In task 3, which merged with the debriefing session, subjects were asked to supplement/modify/expand/comment on the design in new ways. Qualitative data were collected about how well people were informed as to discuss the design, whether - or the extent to which - new design ideas were triggered, etc.

<table>
<thead>
<tr>
<th>Document</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem statement</td>
<td>Requirements</td>
</tr>
<tr>
<td>Design product</td>
<td>Design, elements/objects</td>
</tr>
<tr>
<td>Issue prioritisation</td>
<td>Assesments</td>
</tr>
<tr>
<td>Design rationale</td>
<td>Options</td>
</tr>
</tbody>
</table>

Table 7a: Rhetorical elements included in the experimental documents. Elements towards the upper part of the table tend to be more static, as parts of the final product and its evaluation, whereas elements towards the lower end of the table tend to be more dynamic in that they reflect problems/issues raised during the design process.
7.4.4 Procedure

Training session
In the training session, participants were given a tutorial on DR using QOC. That included an introduction to QOC-based DR, followed by a presentation of a few examples inspired from the IBIS manual [89], followed by a set of short exercises. This tutorial was significantly more elaborating, learning the lessons from experiment #1.

Randomisation procedure
An evens/odds randomisation procedure was used to assign subjects to conditions. Subjects randomly picked one of four numbers, the parity of the chosen number determined their group.

Rehearsal
In this session, subjects were taken through a set of similar (to the main one) tasks and explained what it is requested of them. They were then asked to briefly try out for themselves, and the experimenter commented on their answers and made sure the procedure was clear to them.

Practical session
There were three problem solving sub-tasks for each task/problem, as explained above. In this session, participants were handed the two tasks with a 5 minute break in between. The session was videotaped for the purpose of obtaining response time measurements and a full picture of the qualitative information on task 3.

7.4.5 Scoring conventions

The judgement of answers for correctness is always open in such a task, due to its ill-definition. Responses with an identical or close meaning to the original are accepted as correct, using the experimenter’s judgement.

An expert designer has also agreed to provide an independent marking on the designs. Where major differences exist in the two markings, the independent marker’s ratings will be kept.
7.5 Results

7.5.1 Training session

The training session was more effective compared with the one followed in experiment #1. That was basically as it was longer and thorougher. The main observation was the tendency of subjects to read a design rationale fragment in a designated way, inadvertently giving it a formal character, not originally meant to be there.

7.5.2 Practical session

In task 1, there was a case where a subject (S2) would provide a high rate of correct answers in a very short period of time. That was under the no-DR condition. Both the subjects that were under the DR condition had a fairly low progression rate. One of them (S3) had a percentage of correct answers as high as S2, although the other one (S1) had a lower percentage. The difference in progression rate could possibly be reduced by introducing a placebo document, in order to balance the time spent on the one extra document under the DR condition. I would expect DR comprehension overheads to manifest themselves but certainly hope and try to outbalance them in terms of the performance on correct answers.

In task 2, there was no certain evidence of DR effect. It looks like once the current subjects determined the requirements and the objectives they imply, they expanded the design in terms of what they had completed out of task 1. They very seldom looked back to review the documentation, and that was mainly the problem statement rather than the DR material.

In task 3, due to the relaxed and informal style in which it took place, all designers engaged discussion with the experimenter. Subjects on the non-DR condition raised more methodological issues and elaborated on what they had done, whereas on the DR condition, the relevance of the DR material to the design dominated the discussion, giving mixed feedback. DR subjects did seem to elaborate more on the process they had followed, but at this point it is hard to say whether they had more or better ideas than the other group.
7.6 Discussion, related and future work

From the current experience with this quasi test, I am putting forward a set of criteria for design 'contextual' documentation to be communicative to a level in which transfer of expertise can be facilitated through purposeful reading of it. Those criteria include:

(a) **information theory analysis** that suggests that the information that the provided DR documentation contains the right pieces of information - w.r.t. expertise share - on the decisions that were made that relate to the given design objects

(b) **cognitive fitness** of all the pieces of the given documentation

(c) **appropriateness**, i.e. making sure that the DR material thematically corresponds to the current design question

(d) **ephemerality**, i.e. ensuring that the DR documentation is at the right state to reflect the working design version

It has to be noted that although this experiment provides evidence of the usefulness of design rationale documentation as a design supplement, it does not solve the well-documented problems associated with software documentation. Acceptance of documentation is usually very low in industry and this is an existing current problem.

Although there is much research on design reuse, e.g. [3], etc., no proper investigation has been conducted of the role of DR in this purpose. Ramesh's work [75] comes close to this one in the sense that it does view DR as a knowledge facilitator, although the context which ideas are tried out is fairly different - in fact no testing of the central question of benefit is done.

Such hypotheses although fundamental, would boost the role of DR in software documentation and as a basis for process-based and information-richness-based documentation and expertise transfer facilitator.

Due to its nature, this experiment has not produced conclusive results. However, I am confident that the existing results point us to promising directions.

**Related issues**

Is the strength of DR in that it provides expertise/knowledge in the form of contextual argumentation or is it that the «discussion» that it can create? In a similar note, engagement in discussion, either with oneself or with others is equally important in systems design, or is it that the results of reading a previous discussion are equally important? Is the potential of DR in prompting such a discussion?
Future work

Work is underway to include IBIS in the LOUIS system (see chapter 5) and perform the main experiment on the computer. The advantages will be crucial, as this will enable us to perform many tests at the same time, while minimising any experimenter effects and controlling noise factors more effectively.
7.7 Chapter summary

In this chapter, I described a quasi-experimental study that investigated the information needs of a software designer who tries to re-use existing documentation. In particular, pieces of information were classified as processional or dynamic and static on which dynamic knowledge has to be inferred, and there were tested: (a) the effect of design rationale documentation on facilitating such inferences, and (b) whether the «extra» contextual knowledge it provides is beneficial to the progression of the design. Although no conclusive results are available, indications are that design rationale provides an important structure for contextual design knowledge, often lost in the process for delivering a product on time.
CHAPTER 8

CONCLUSIONS AND FUTURE WORK

Table of contents

8.1 GENERAL CONCLUSIONS ........................................204
8.2 THE THESIS IN RETROSPECT ..........................204
  8.2.1 PART A ....................................................205
  8.2.2 PART B ....................................................206
  8.2.3 PART C ....................................................207
8.3 FUTURE WORK ..................................................208
8.1 General conclusions

The aim of this thesis was to investigate the role of design rationale in software design - towards that aim, a lot of empirical evidence suggests the properties that make DR useful in such a context, the types of task, as well as the appropriate representation. Certain suggestions are also made as to how we can employ DR-based research results into practical design.

Design rationale research is extended to high-level software design by identifying properties of argumentative theories that can be useful in software design tasks. From a theoretical perspective, this research proposes an empirically-derived model of the decision making process in a typical software design task, and illustrates how decision making relates to salient points in the process - points for knowledge discovery. From a practical perspective, it proposes a set of ways of transferring expertise using inherent features of DR methods.

More specifically, in this thesis it has been demonstrated how DR can be used to transfer expertise across software designers to assist certain task types which are prone to uncertainties and breakdowns by putting forward the conditions under which DR can be most useful in that role. I have presented the results of a critical analysis of the design process from an argumentation viewpoint, after empirically investigating the use of DR in several stages in the life cycle. In terms of design construction, the reasoning process has been modelled with respect to breakdowns. In terms of evaluation, I have performed two experiments that assess the appropriateness of certain DR attributes for the purpose of making it re-usable. The implications of the results of this work for (a) software design education (b) design of support tools, and (c) design methodologies are substantial.

8.2 The thesis in retrospect

Let us evaluate the work and see the extent to which the research objectives have been met and what future work is needed to fill in any aspect of the objectives which has not been completely achieved. The evaluation is done on a per-part basis. As a reminder, two plans (or 'sub-objectives') were derived out of the main thesis objective, with respect to the design of software. (a) Deeper understanding of the process and (b) provision of information that can lead to the improvement of that process. It is now seen how each part of the thesis has done in terms of completing those two plans.
8.2.1 Part A

Initially, in search of a starting point to the investigation, we performed a survey of the literature that refers to 'alternative' ways of looking at the software design process. In terms of the underlying philosophy, those other approaches are akin to theories of argumentative reasoning as well as design theories in general, whereas in terms of methodology, they are akin to Experimental Psychology. What we needed at this point was a good understanding of the software process. Considering that the software design process is multi-faceted, we looked for parts of the process that are interesting and can be used as milestones on which to base any comparisons for the purpose of process improvement.

The design processes that were identified were analysed in three levels, and at the same time a set of software tools that support design was put under scrutiny. The results of these two surveys were cross-referenced in the Software Design Support Framework (SDSF). One of the main outcomes of this approach was the identification of a number of processes that although prominent in design behaviour, were inadequately supported by tools.

In parallel, we set out a context for the study of the software design process. Despite the high number of studies of software design, it is felt that the current way the subject is approached is not systematic, in the lack of a framework that puts all pieces of work together and makes results comparable. A typical related problem one faces when studying the bibliography is that most studies on the software front refer to programming rather than high-level design. The resolution to those problems is achieved by defining a set of factors, the software design (context) factors, the instantiation of which restricts the scope and focuses attention on particular aspects of the design process.

Our focus turned on breakdowns, difficulties faced by designers that arise from the limitations of human cognition and lack of (mainly domain) knowledge, recovery from which can result to the discovery of a new viewpoint and resolution of the problem at hand. Our investigation of expertise transfer phenomena started off from that point. The next step was to find out some sort of platform on which the knowledge itself would sit upon. Investigating the literature for the types of knowledge that cause breakdowns and in general make a difference in design performance, it was identified that argumentation is an interesting prospect.

Although several claims have been made in the literature about the relationship between design rationale and breakdowns and their related behaviour, no thorough investigation of that relationship had been made so far. Our objective at this point was to operationalise that relationship in the right task and test the potential for expertise transfer that it may convey.
In this part of the thesis, it is felt that our first plan-objective has been largely met. A good command of the current bibliography has been reached at and a fairly good and multi-faceted understanding of the design process — as that is described by researchers from different disciplines — has been achieved. Nevertheless, there were areas for future investigation, as mentioned in chapter 2. These mainly include the elaboration and continuous update of the SDSF framework and are fully expanded upon in the 'future work' section — (8.3).

8.2.2 Part B

In the next part of the research, we decided to spend a considerable amount of time in understanding design practice as it actually happens. The SEDRES project was just the right opportunity in that line and it yielded two main points:

• A Generic Process Model (GPM) of the systems engineering life cycle as well as its associated notation
• A mechanism of evaluating the system engineering process from a set of interesting perspectives including breakdown behaviour.

Following that, we performed an exploratory study of the relationship between breakdowns and decision making in the behaviour of five software designers that were PhD candidates. Protocol analysis revealed a plan-based decision making, and restricted and implicit design argumentation. In some cases, methodological reasoning dictated the decision making process whereas in other cases strategic planning was more exploratory and based on generic heuristics. In both cases, though, breakdowns and decision making were related and did so in certain patterns, as illustrated in chapter 4. In brief, the study yielded the following:

(a) points in the process where expertise was needed (backed by Guindon's study [37]),
(b) instances of how recovery was then approached, and
(c) knowledge requirements for the purpose of recovery, from the vision of providing that knowledge through DR languages.

In this part, that first plan-objective is further met, as our view of the design process has been complemented with a new facet which stems out of empirical investigations. On the one hand, our experience with the SEDRES project has provided general knowledge of the field and the background to appreciate real design — the types of problems designers face in industry and the research issues that associate to them. On the other hand, the protocol analysis study has increased our understanding of design behaviour and how it is affected by individual differences. In addition, a basis of how we can make use of argumentative reasoning to increase performance with specific reference to breakdown behaviour is set up. These two parts have completed our picture of the design process to a satisfactory level. We can thus be fairly happy with the degree of fulfilment of our first plan that relates to our objective and feel
confident in moving on to the second plan, which is the provision of information that can lead to the improvement of the software design process.

8.2.3 Part C

In the next step (chapter 5), we investigated the appropriateness of current DR languages for that purpose. Part of that work involved the development of LOUIS, a QOC-based tool for DR management. Our investigation concluded that certain descendants of the IBIS methodology were appropriate for certain design tasks. For user interface design tasks, QOC was appropriate, whereas for the design of the 'core' (i.e. non-interactive part) of systems, the QAR and Potts and Bruns approaches were more suitable.

To drive the investigation to its final point, two experiments were conducted which both tested the suitability of certain aspects of DR for the transfer of knowledge among designers. The first experiment, which compared representational and structural paradigms of design languages, demonstrated among other points that:

- design rationale due to its generic semantics is easily comprehended and argued upon by novice analysts
- graphical design rationale is was understood more quickly than its tabular counterpart, which is in turn better understood by its narrative counterpart, although none of those differences was statistically significant in our experiment. An exception is semantics tasks - i.e. where subjects had to make good sense of the given argumentation and possibly make inferences - where tabular notation is grasped significantly more quickly than its graphical and narrative counterparts.
- graphical form was the subjects' favourite notational option for design rationale, with tabular and narrative following. The same result was obtained in terms of the subjects' votes on the categories of intuitiveness, suitability and effectiveness.
- an options-based argumentative structure suits novices better than a criteria-based one. An options-based structure leads the reader from options to criteria whereas the reverse is true in a criteria-based structure.

The second experiment indicates that design rationale can provide the context that a set of design diagrams need in order to be understood to a larger extent. Subjects were able to engage themselves in comprehension and ‘silently’ participate in a design discussion, picking up from the points the previous designer(s) left. That act increased their appreciation of the process that was followed in order for the diagrams to be produced, and this potentially increased their design expertise by presenting them to a (most probably) different way of approaching the problem. In addition, they were able to detect design changes that were caused by the implementation of certain design decisions. That demonstrated a considerable degree of compatibility between design and design rationale documents.
It is felt that this part has also successfully contributed to the satisfaction of the thesis objectives, to a realistic extent. More specifically, the significant contribution of chapter 6 with its novel theory of comprehension of design material, has driven the point of this work home. That experiment established the role of task nature in DR comprehension, in a manner which unifies theories of code comprehension, and 'cognitive fitness' in decision making. Lastly, the last experiment confirmed that there are circumstances under which DR can be re-used effectively and increase designers' performance. The importance of that experiment – apart from its confirmatory character – lies on the fact that it forms a new research agenda as it creates a new experimental framework, to the one that was used in the thesis. Through that new framework we can study the communicability of design material, through the richness of information it conveys. The issue is elaborated in section (8.3). Certain areas in this part could have been researched further. In particular, in chapter 5, more thorough evaluation of the existing approaches to DR can be foreseen. That is further elaborated in section (8.3).

This thesis makes several points about the usefulness of DR in software design, both in terms of its position in problem-solving assistance and also as a context for implicit learning. Nonetheless, there have been identified a host of points that need to be addressed in order to make DR readily effective and useful in software design practice. The core of the recommendations made here, stems from the trend to produce tailor-made design rationales. I advocate the need for a formal classification of problems so that their associated rationales are tailor-made in a systematic rather than arbitrary way. That would enable cross-domain analogies to be drawn more easily and thus re-use of rationale facilitated.

8.3 Future work

The elaboration and continuous update of the Software Design Support Framework is an area that is interesting to pursuit. Nevertheless, it is seen more in a long-term basis. New empirical data on the software design process are very common. As hardware and software technologies evolve and new ones emerge, the way people design software changes. A smaller proportion of designers than ever before build systems from nothing, but they try to make use of existing software. Integration of components, cross-platform development and distributed software engineering are new practices that need understanding and modelling in terms of human cognition. These new ways of work can be depicted on the SDSF, to maintain its global character and to spot processes that need to be looked at more closely.

As explained in chapter 5, the LOUIS software was not actually used as part of the proper thesis experiments. Under the right circumstances, the LOUIS tool can be further developed to include a fully functional environment of design rationale management. By incorporating a recorder-reviewer of actual system use, LOUIS would provide an experimental platform on which one could work. Multi-user and distributed processing facilities can also be added fairly

208
easily, given the capabilities of Tcl. That is a possibility which can considerably enhance the prospect of communication in design, and open new issues of collaboration, not investigated in the thesis. The prospect of using LOUIS in experiments is very large, considering that not much empirical work has been reported on design at work on tools. The extent to which tools affect the way people work as well as on which properties they do so, is an area with much to offer. In addition, using software to perform experiments adds a de facto rigidity to the experimental procedure and eliminates experimenter effects. These two elements ensure experimental results of higher accuracy and validity.

As mentioned in the previous section, the appropriateness of DR languages for certain software design tasks is a theme that would have been worked on further, given time. One approach would be to produce a conceptual framework similar to SDSF, which matches human design processes and DR-based tools, and reveals strengths and weaknesses. From the perspective of keeping the framework open to elaboration from empiricism, one could envisage the theoretical part of such a framework (Design activities) to include a complete coding scheme on which the results of different protocol analyses or other types of exploratory studies can be added upon. That way we can have a global 'purely empirical' view of the design process and its support. This whole research theme stems from the issue of adequate classification of design problems. Although initial work on «wicked» problems is a good start and there are certainly indications that DR can be useful in dealing with them, it is our view that without further work on (a) the further classification of problems in general, in terms of their structural characteristics, and (b) match of that classification with sound descriptions of the design tasks throughout the software life-cycle, DR’s role in design cannot progress much further. This whole theoretical approach constitutes from the fact that not much is known as to which (albeit whether) types of argumentation fit which types of design tasks. In fact, our study is a first step to that direction. On a similar note, empirical work on software design does not follow typology of problem tasks which often makes disparate research results difficult to compare. Such a typology would alleviate some of the difficulties that relate to making DR part of everyday design life.

Another proposal that might also be interesting would a library of DR which should be backed by a good classification of problems (see above paragraph) - or else browsing criteria would be arbitrary – which will include cross-domain similarities etc. It would also be interesting to see how the ‘important’ design points to be identified in experiment #2 relate to the library. Considering guided exploration of problem isomorphs would provide a large prospect for the employment of computer-based learning techniques in the design space world. Another interesting point is that problem isomorphs is probably a better basis for the measure of expertise, in the place of formal qualification descriptions.
Lastly, chapter 7 opens up a new area of investigation by focusing on the information richness of design notations. Experiment #2 as it stands, provides a good starting design of experiments that can test the readiness of certain notations and methodological operators to cope with the dynamic character of an evolving specification. The capability of certain types of design documentation to facilitate inferences and to convey context, are very important assets of their communicability. One can envisage the use of the design of experiment #2 to implement a benchmark of DR approaches, a test of combinations of design languages - both essential and complementary - and so forth.
References


[17] CSCA (Computer Supported Co-operative Argumentation) discussion list archives at cscas-requesert@maillbase.ac.uk

211


[45] Iliadis, G P., Candy, L. and Edmonds, E., Even software designers are creative: implications for computer support environments, Working paper, LUTCHI Research Centre, Loughborough University, 1996.


[65] NASA Langley formal methods program at the location http://shemesh.larc.nasa.gov/cgi-bin/fm.cgi?+s1#s1


[79] SEDRES project, at LUTCHI home page, http://www.lboro.ac.uk/departments/co/research_groups/sedres.html


[90] Tcl archive, at www.neosoft.com/tcl/


Appendix A

Documents related to the SEDRES project

SEDRES GLOSSARY ................................................................. 218
STRUCTURED INTERVIEW .......................................................... 224
FEEDBACK FORM ..................................................................... 235
THE GENERIC PROCESS MODEL NOTATION TUTORIAL............ 237
COMPANIES PROCESS MODELS ................................................ 246
MAJOR EVALUATION DATA COLLECTION FORMS ...................... 251
TRANSFORM REPORT EXAMPLES .............................................. 256
SEDRES Glossary

Alphabetical List of defined Terms

Action
Any step performed by a user or system during a usability evaluation session. Not specific to the task in hand.

Allocated Use Process
A process specific to a Use Scenario explicitly identifying roles adopted by engineers and design tools.

Architectural Design
The process of defining a collection of hardware and software components and their interfaces to establish a framework for the development of a system. [IEEE Std 729-1983]

Baseline
A specification or product that has been formally reviewed and agreed upon, that thereafter serves as a basis for further development, and that can be changed only through formal change control procedures (IEEE Std 729-1983)

Behaviour
The dynamic properties of a function or a system.

CALS
“Continuous Acquisition and Life-Cycle Support” is an overall concept developed by the DOD which allows the customer to be closely involved throughout the development and lifetime of the delivered system.
The STEP data exchange standard (which is being considered for adoption by SEDRES) is one of the enabling technologies which will support this concept.

‘Capability’ (SEDRES Specific in the context of Data Exchange)
This term has been used within SEDRES specifically to refer to the data exchange capability at different stages of the SEDRES project. Two Capabilities for data exchange will be developed within the SEDRES timescales which are referred to as ‘Capability/1’ and ‘Capability/2’. The implementation options for Capability/1 &/2 will be defined in Task 4.2.

1 “...of a system...” replaces “...computer system” for the definition provided in Ref 1 to make it more appropriate for the SEDRES target applications.
Change Control
The process of ensuring that changes to elementary constituents of a large compound item (for example requirement, design or implementation) are made with proper consideration for the impact that the change in one item may have on the others. Typically this includes consideration of the impact of a possible change and identifying a number of items that have to be changed together to ensure consistency. (Ref. 1)

Concurrent Engineering
A method of working in which successive design phases are performed at the same time in order to reduce overall cost and development timescales.

Configuration Control
A process of ensuring consistency among the many parts of a multi-part item (for example requirement, design or implementation, or the contract or project plan). There may be several variants being developed concurrently, for example for several different sites or target computers. A total configuration may include items of different categories, for example software, hardware, and documentation. Configuration control often includes change control as an important element. (Ref. 1)

Context
All factors that affect the usability of a product, but not the product features themselves.

Context Analysis
Factors designed to support the evaluation of a product's usability in realistic situations.

COTS Tools
“Commercial Off The Shelf Tools” are the design tools used (or being considered for use) by the SEDRES partners companies. The support of COTS tool suppliers is necessary in order to develop import/export capabilities within these tools which will support the SEDRES data exchange standard.

Design
The process of defining the architecture, components, modules, interface, test approach and data for a software of a system to satisfy specified requirements. (Ref. 1) Also “A design”; the product of a design process.

Design Entity
An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced (IEEE Std 1016-1987); maybe hardware, software or a combination of the two.

Dynamic Functionality
A description of a set of requirements of the design which includes timing and behavioural information.
Effectiveness
A measure derived from Quantity and Quality of measures made.

Efficiency
Effectiveness divided by the input to the task. Input can be measured by effort (human efficiency) or total cost (corporate efficiency). Input may be measured by Task time giving User Efficiency.

Evaluation
The process of determining whether an item or activity meets specified criteria.[DOD-STD-2167A]

Exploitation
The process of obtaining business benefits from a product or improved process.

Function
(1) A specific purpose of an entity or its characteristic action (ANSI) [IEEE Std 1063-1987]
(2) A subprogram that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation, Contrast with subroutine.[IEEE Std 729-1983]

Functional Requirements
The stated required active functionality of a system or product.

Functionality
The existence of certain properties and functions that satisfy stated or implied needs of users. [IEEE P1061/D20, Appendix A]

Generic Process Model
A model of a development process which is sufficiently SEDRES-partner independent to be common across all partners and recognisable to the rest of the Systems Engineering Community

Integration
The process of combining software elements, hardware elements or both into an overall system. [IEEE Std729-1983]

Life-cycle phase
A period of time during development and maintenance that may be characterised by a primary type of activity (such as design or testing) that is being conducted. (Ref.1)

Metric
A well defined means of assessing the degree to which an object possesses a given attribute. (Ref.1)
Model
A model corresponds to some aspect of reality, and features shared in common by the model and that reality indicate what is deemed to be significant by the modeller. There may be several models of the same reality, in which different features are deemed to be significant. (Ref.1)

Performance
The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage. [IEEE610-12-1990] Performance may also apply to aspects such as energy usage or frequency response.

Properties
Frequently termed the 'ilities', a property is a non-functional characteristic of a system or product, such as reliability, usability or safety.

Process
The series of actions or operations used in making or achieving a product. [IEEE Std 1002-1983]

Process Model
A formal description of the general features of a class of processes (Ref.1)

Quality
A measure of the degree to which the task goals represented in the task output have been achieved.

Quantity
A measure of the amount of a task completed by a user. The proportion of the task goals represented in the task output.

Product
Any system or device that is designed to support a task or set of tasks carried out by a user or groups of users.

Requirement
A condition or capability that must be met or possessed by a system or system component to satisfy a contract standard specification or other formally imposed document. (Ref.1)

Specification
A document that prescribes, in a complete, precise, verifiable manner, the requirements, design behaviour, or other characteristics of a system or system component. (Ref.1)
Static Functionality
A description of a design which excludes timing or behavioural information.

STEP
"Standard for the Exchange of Product model data" is the name given to the standard ISO10303 and is the exchange standard which will ultimately be adopted by SEDRES for the exchange of Systems Engineering Design data.

System
An assembly of interacting, active components or elements forming a whole.

System Architecture
The structure and relationship among the components of the system. The system architecture may also include the system's interface with its operational environment. [IEEE Std 729-1983]

System Design
The process of defining the hardware and software architectures, components modules, interfaces and data for a system to satisfy specified system requirements. [IEEE Std 729-1983]

Systems Engineering
Due to the difficulty of defining Systems Engineering, four representative definitions are provided along with their respective sources:

(1) The application of methods and tools to the development and production of (reliable and maintainable) systems. (Ref.1 )

(2) ....the application of scientific and engineering efforts to
- transform an operational need into a description of system performance parameters and a system configuration through the use of an iterative process of definition, synthesis, analysis, design, test and evaluation.
- Integrate related technical parameters and ensure compatibility of all physical, functional and program interfaces in a manner that optimises the total system definition and design.
- Integrate reliability, maintainability, safety, survivability, human engineering and other such factors into the total engineering effort to meet cost, schedule, supportability and technical performance objectives.
[MIL-STD-499A Systems Engineering]

(3) Systems engineering is an interdisciplinary approach to evolve and verify an integrated and optimally balanced set of product and process designs that satisfy user needs and provide information for management decision making. [MIL-STD-499B Systems Engineering]

(4) ...is a hybrid methodology that combines policy analysis, design and management. It aims to ensure that a complex man-made system, selected from the range of options on offer, is the one most likely to satisfy the owner's objectives in the context of long

**Task goal**
The objective of performing the task. The task goal should be clearly stated before task time starts.

**Task output**
The result achieved by the user and system at the end of the task which can be examined to determine how effectively the task goal has been achieved.

**Test and Evaluation**
The process of examining a product to determine whether it satisfies the need for which it was produced.

**Use Process**
A definition of how the Systems Engineering design work is to be performed within a Use Scenario

**Use Scenario**
A predefined application of the data exchange standard which is intended to be representative of a 'real' application. Three Use Scenarios are used in SEDRES for assessing and demonstrating both Capability /1 & /2 of the developing data exchange standard.

**Usability**
The extent to which a product can be used with effectiveness, efficiency and satisfaction by specified users to achieve specified goals in specific environments.

**User**
Another system, human or physical, which interacts, either as client or as server with the considered system. (Ref. 1)

**Validation**
The process of evaluating a product at the end of the development process to assure compliance with its requirements (Ref. 1)

**Verification**
The process of determining whether or not the products of a given phase of the development life cycle fulfil the requirements established during the previous phase.

**Version**
A controlled item with a defined set of functional capabilities. As functional capabilities are added to, modified within, or deleted from a controlled item, the item is given a different version identifier. [P1074/D4 - Aug 1989]
Structured Interview

Contents

BRIEFING

A. Objectives
B. Confidentiality
C. Methods

INTRODUCTORY QUESTIONS

A. Details of interviewee

PART I. THE OVERALL (SYSTEMS ENGINEERING) PROCESS ("BIG PICTURE")

A. Reference to a project
   - Product Design Specification and Assumptions List

PART II. THE SYSTEMS ENGINEERING PROCESS IN THE CONTEXT OF USE SCENARIO 1

A. Customer/User Requirements
B. Concept Design
C. Detail Design Stage
D. Overlap and interrelationship between stages

PART III. COMPUTER SYSTEM AND HUMAN SUPPORT

A. Design Tools and Their Users
B. The Data Exchange Issue
BRIEFING

A. Objectives

The aim of this interview is to acquire a more complete and structured set of data about the design process at [BAe, SAAB] as an example of aerospace industrial practice. The aim of the study is three fold:-
(i) the way the systems engineering process operates within the organisation.
(ii) the role and activities of individuals and teams - both formal and informal activities
(iii) the use of design tools in the process, their contribution and limitations.

B. Confidentiality

All information is strictly confidential and will be reported anonymously. Agreement will be sought where material is needed for publication.

C. Methods

The interview is a structured one with scope for open discussion.
Interview data will be recorded (with permission) and transcribed.
The transcription will be provided to the interviewee and feedback sought.
An opportunity to expand upon or modify any part of the discussion will be provided. It is hoped that this interview will provide the basis for an on-going dialogue between the two parties.
INTRODUCTORY QUESTIONS

A. Details of interviewee

1. Name
2. Company
3. Position in Company
4. To whom do you report or liaise with? Who reports to you?
5. What are your main responsibilities?
6. What are the main projects you have been involved in the last two years? What was your contribution to them?
7. I am now going to show you four statements that express some views of design. Can you tell me how much you agree or disagree with them and to what extent they reflect the way in which engineering design is thought about in this company.
I would now like you to consider one project which you have worked on recently and with which you are quite familiar. If possible, consider a project that includes scenarios similar to any of the Use Scenarios (perhaps US1, for the moment) and imagine the process that overlooks it. Can you tell me about it? In particular, how did you go about designing, when working with others - talking discussions and when you work alone?

1. What was the product?

2. Was it typical of the design process normally undertaken by the company?

3. Was it an update of an existing product, a facelift or a completely new venture?

4. What was your role in the design?

5. At which stage did you become involved?

6. At which stage did you cease to be involved?

7. Were there any particular problems with this project (e.g. liaison, information access, team or group mix, hangups?) people problems/ machine problems?

8. Was there a Product Design Specification or equivalent document? (see next page)
Product Design Specification and Assumptions List

1. Can you describe what sort of things are contained in this document? The following table might provide some help.

**Issues included in the PDS or equivalent document**
(please tick or state as appropriate)

<table>
<thead>
<tr>
<th>Aesthetics</th>
<th>Packing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Specification</td>
<td>Maintenance</td>
</tr>
<tr>
<td>Ergonomics</td>
<td>Competition</td>
</tr>
<tr>
<td>Life in service</td>
<td>Quality</td>
</tr>
<tr>
<td>Reliability</td>
<td>Performance</td>
</tr>
<tr>
<td>Product cost</td>
<td>Patents</td>
</tr>
<tr>
<td>Timescale</td>
<td>Environment</td>
</tr>
<tr>
<td>Customer</td>
<td>Testing</td>
</tr>
<tr>
<td>Processes</td>
<td>Safety</td>
</tr>
<tr>
<td>Size</td>
<td>Company constraints</td>
</tr>
<tr>
<td>Shipping</td>
<td>Politics</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>Product life span</td>
</tr>
<tr>
<td>Market constraints</td>
<td>Materials</td>
</tr>
<tr>
<td>Weight</td>
<td>Quantity</td>
</tr>
</tbody>
</table>

Others...(please specify):
2. Are ergonomic considerations usually found in this document?

3. Do you perform any type of situational analysis e.g. specify worst and best expected cases?

4. Do you know who had suggested the constraints, and what was the rationale behind them?

5. Was there a design log kept of decisions?

6. At what stage is a PDS produced?

7. To what extent does the assumptions list guide activities?

8. Who has the final say regarding what it says?

Did the finished design meet all the specifications as far as you are aware?

9. How many people were involved in the project?
- by stages: problem definition, concept design, solution generation
  : detailed design, marketing etc.

10. Was the team put together especially for this project?
    Had the team members worked together before?

11. Did the membership stay the same throughout the early stages?

12. Who has overall responsibility for the project?
    What happens when it leaves concept design?

13. Are there any checkpoints on the process which management requires reporting planning etc.? How does it work in terms of infra- and inter-team communication?

14. As people come into the project, do they have any problems getting to grips with the design? How do they understand where the design is at? (by history reports, diagrams, verbal reports?)

15. Are the teams inter-disciplinary? Is this company policy?

16. How does the core design team liaise with other departments, especially on day to day issues (e.g. meetings schedules, telephone, email. etc.).
PART II. THE SYSTEMS ENGINEERING PROCESS IN THE CONTEXT OF USE SCENARIO 1

A. Customer/User Requirements

Once you have been passed a brief, how do you start firming up on it, especially with regard to customer requirements and ergonomics?

1. Is any one person responsible for looking at the customer/user needs.

2. How do you check that the design actually meets the customer needs? (e.g. design review)...at the concept stage and as the product develops.

3. Do you have any tools or computer systems which help you consider the customer needs during the whole design process?

4. At what stage of design are user interface issues considered? Do you think they should be considered separately from the functionality, and at which stage?

5. Do you think that functionality and user interface complexity should be dealt with from different partners? If yes, could you sketch the implications for information exchange?

6. What information do you use to help you with ergonomic issues (man-modelling systems, mannequins, journals, style guides).

7. Is ergonomics one of the first things to be considered?
   - Does it have its own specification?
   - Does it ever lead to disagreements?

8. Can you find all the information you need?

9. To what extent do the ergonomic issues in the specification get ignored further down the line?

10. When you think about your product users, which people come to mind (e.g. pilots, clients etc.)
B. Concept Design

1. Can you tell me a bit about the sort of work you do in this stage?

2. Am I right in thinking everyone in the team is responsible for a particular area? (aesthetics, styling, interiors etc.).

3. How do these areas relate to each other? How do you know that your bit will fit in with someone else's?

4. How is the team co-ordinated (informally, working in same office space, formal meetings and reviews)?

5. Do you work mainly with drawings?

6. Do you have a design log/history to keep track of decisions? What about the associated rationales?

7. What are the main influences on concept design?

8. How many concepts are normally considered? Are they joint efforts or individually generated?

9. Do you ever contract out at this stage?

10. How do you bring the design activity to an end? What types of criteria are used? Is this point also a management "checkpoint"? (see question #13 at "PART A: Overall Process")

11. Is agreement reached about a concept through meetings? If so, who is involved? What is the presentation media?

12. Is the final decision made using any of the following methods? Decision trees, decision matrices, concept selection matrix, product planning matrix (QFD) other?

13. How do you communicate your ideas to other departments? (drawings, presentations, whiteboards, reports etc.)
C. Detail Design Stage

1. When does detail design start?
   (as a separate stage after formation of concepts, or in parallel?)

2. How much detailed design do you carry out at the concept design stage?

3. Do you use formal methods at this stage (QFD, Taguchi etc.)

D. Overlap and interrelationship between stages

1. How much revisiting takes place? e.g. between problem specification, solution generation and concept design.

2. What is the role of prototyping?

3. In the early stages how much effort goes into bringing the team members up to status with a particular product or new information?

4. Is the solution generation preceded by extensive consideration of alternatives?

5. Does the team move immediately into considering a particular solution?

6. How often is the solution based upon an existing part or product?

7. How often do conflicts arise?

8. How are conflicts resolved?

9. Could you mention any typical errors occurring in the communication between teams and/or partners? Also, is this communication (a) fast enough, (b) secure?
PART III. Computer System and Human Support

A. Design Tools and Their Users

1. What computer or paper-based tools do you use in throughout the design process as described above and by whom?

2. What are their strengths and limitations?

3. Would a computer support system that helped you consider product user needs be useful to you? If so when would you use it?

4. Do you have a means of producing a design history? (by computer or otherwise).

5. If a tool was developed for use at the early stages of design what systems would it have to fit in with?

6. What in your view are the main problems with the existing computer systems that are in use in the company?

7. Are you aware of any Integrated Project Support Environments (IPSEs)? Do you think that a single tool would be able to cope with the whole product lifecycle, or do you think that the existence of a variety of different tools is inevitable?

8. Do you think that the list of tools selected for this exercise have to be “state-of-the-art”?

9. In your opinion, are there any other types of tools that should be included in the exercise which are not included at the moment? What is the selection procedure, and are there things like criteria for appropriateness/ special priorities?
B. The Data Exchange Issue

1. What types of data/knowledge is typically exchanged and at which stages?

2. What types of mismatches do humans / tools try to bridge during Use Scenario 1? Here is a starting list which you are asked to elaborate:

<table>
<thead>
<tr>
<th>Mismatch type</th>
<th>Process/ work situation that causes it</th>
<th>Involved tools</th>
<th>Involved humans</th>
<th>Further comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level of abstraction</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thank you very much for your time!
Feedback form

(1) Scope of discussion in terms of LUTCHI role in SEDRES

Please add your comments on the list of points below which relate to the SEDRES Generic Process Model in preparation.

(2) Attributes of the generic process model

The generic process model should be:

• readable

• relatively simple

• aesthetically acceptable

• informative

• flexible

• representative of the concurrent engineering process

(3) The submitted models

• Most make use of “baselines” e.g. functional, allocated and product but not at the same number/extent

• Most of them do not show concurrency aspects

• Most include definition, design/development and testing/integration phases but do not necessarily follow the same terminology

• Some emphasize software development as the major component while others do not

• Underestimation of role of humans and tools (?)
(4) **Perspectives to be represented in the model**

- Functional
  - static (e.g. De Marco)
  - behavioural (sequencing, causality, process states etc.)

- Non-functional: redundancy, adaptability, timescale, efficiency etc.

- Allocation: people, tools, special extent (proximity), resource usage

(5) "Important aspects that impact the data exchange definition"

Identification of such aspects and associated activities/processes in order to elaborate the "baseline" view:

- Requirement evolvement Vs Assumptions management

Possible examples:
- Inference of constraints
- Addition of new requirements
- Consistency check
- Abstraction of key points and prioritization
- Browsing/editing/querying facilities

- Feedbacks
- Data consistency management
- Traceability
- Introduction of existing entities
- Data access control
- Industrial responsibilities and properties
- Bridge of (a) the abstraction level
  (b) different perspectives/paradigms
- Team environment
- Terminology
- Interpretation of design information
The Generic Process Model Notation Tutorial

A rectangle (notational object #1) is used to represent system architectural elements. Any physical system element belongs to this category, from products to systems, subsystems, components, subcomponents, parts etc.. The compositional relationships of these elements i.e. which subelements is each element comprised of, are denoted using a continuous straight line (notational object #2). For example, in the right hand part of figures 2a, 2b under the System architectural perspective, we can see a hypothetical avionics product which consists of a number of systems, i.e. System1, System2, etc., System1 consists of two subsystems: Subsystem1.1 and Subsystem1.2 which in turn may consist of a number of components. An example of the use of the rectangle is shown in the following figure.

![Diagram](image)

Figure A1-2. The system architecture of the equipment which US1 is concerned with.
A round-edge rectangle (notational object #3) is used to represent a Life Cycle Process [2] (LCP), i.e. an instance of a generic process that takes place during a stage in the development of system products. Frequently a stage will correspond to a contractual milestone for the development of the whole product. What tends to happen in the examples of this document is that company LCPs that directly correspond to LCPs of the generic model are drawn in solid lines, whereas LCPs that are “proprietary” i.e. occur only in the companies’s own models are drawn in dotted lines. Reference to SAAB model. An example of the use of the round-edge rectangle is shown in the following figure which shows a process fragment from the Aerospatiale process model.

![Figure A1-3a. The original process fragment.](image)

![Figure A1-3b. The same process fragment using notational object #3.](image)
An ellipse (notational object #4) is used to denote an element of a Systems Engineering Process [2] (SEP). SEP is an instance of a generic process that is performed at each of the system development levels (system, subsystem, component). An example of the use of the eclipse is shown in the following figure which illustrates a process fragment of the Alenia process model.

![Diagram of process fragment](image)

**Figure A1-4a.** The original process fragment.

![Diagram of process fragment](image)

**Figure A1-4b.** The same process fragment using notational object #4.
A circle (notational object #5) represents a Use Scenario Process (USP), i.e. an instance of a SEDRES-specific application process derived from a certain case study. The use of this symbol is directly derived from [4], where a circle ("bubble") is used to represent a function in a data flow diagram. For example, in figure A1-5, the process BA4 - Refine system level design is performed potentially by BAe as part of Use Scenario 1. It can clearly be seen that this is part of the SEP component which is called Functional Analysis and belongs to the System Definition LCP. Similarly, the SEP component "Requirements Analysis" which is part of the LCP named "Sub-system Preliminary Design" includes (in this example) only one USP which is called "Develop study material (SA1)" and is possibly performed by SAAB (hence "SA"). Therefore the hierarchy of processes is quite evident.

![Diagram of process hierarchy](image)

Figure A1-5. A model of the process in US1 - functional view.
The open rectangle (notational objects #6) denotes a data store in the same way as in a conventional dataflow diagram [4]. The following diagram presents a part of the description of the US1 process "Perform US1 design work and evaluate".

Figure A1-6. Fragment of a US1 process.

A rectangle bar (notational object #7) represents the duration of a process (USP in principle) and is used exclusively on Cartesian axes, to avoid potential confusion with notational object #1 which has a totally different meaning. For example, in figure A1-7 below, we can see that USP "BA1" precedes BA2 and partly overlaps with SA1, and so on. Therefore we are talking about relative duration of process.

Figure A1-7. The process in US1 - timeline view.
A continuous-line arrow (notational object #8) denotes any type of data or information flow as in popular notations like DFDs etc [4]. An example of the use of the continuous arrowed line is shown in the following figure.

![Diagram showing data flow between customer requirements and hardware/software specification](image)

Figure A1-8. US3 process fragment: the transfer of data between the prime contractor and an equipment subcontractor.
A narrowly-dotted line (notational object #9) denotes a designated area which is usually but not necessarily accompanied by a label that justifies its purpose. An example of the use of the narrowly-dotted line on a fragment from the SAAB process model is shown in the following figures.

Figure A1-9a. The original process fragment.

Figure A1-9b. The same process fragment using notational object #9.
An asterisk (notational object #10) means that the certain object on which it is attached, might have multiple instances. For example, we can have a situation where a system consists of a number of subsystems, say \( n \), \((n \) is a small positive integer\). Therefore, \( n \) instances of the virtual LCP "Subsystem Definition" are being performed, one for each subsystem. The choice of the asterisk was made in order to save a potentially three-dimensional representation of the above situation and also keep the V-shape of a "typical" life cycle. An example of the use of the asterisk on a fragment from the Aerospatiale process model is shown in the following figures.

![Diagram](image-url)  
Figure A1-9a. The original process fragment.

![Diagram](image-url)  
Figure A1-9b. The same process fragment using notational object #10.
The T3.3-specific notation is introduced to support the need to represent the role of humans and tools in the process.

Actors of processes are basically the agents that perform the underlying functions. In the context of T3.3/USPs, it has to be clear who is doing what. An actor might be a computer tool (actor #1 - thick circle) performing an automatic progress on a piece of design, a human using tool (actor #2 - double circle), i.e. a designer modifying a design, an individual (actor #3 - darkly shaded circle) offering some feedback (e.g. verbally or on paper) to another designer, or a human team (actor #4 - darkly shaded double circle) meeting to commonly make some design decisions. The annotation is initially adopted from [3]. Consequently, what is exchanged between actors, tools etc. has to be differentiated. Different types of arrow lines will be used as variations of notational object #8 (arrowed line) to show the different exchange possibilities. It is preferred not to elaborate on this point further as it is subject to change in T3.3.
1. Aerospatiale
2. Analysis
PROCESS MODEL DATA FLOW DIAGRAM - PART 1

3. British Aerospace
3. British Aerospace
System Development Model

Operational Baseline  
Functional Baseline  
Allocated Baseline  
(SUB)System Product Baseline  
Product Baseline

Customer  
SSR  
SDR

Customer  
SSR

SSR = System Requirements Review  
SDR = System Design Review  
SSB = Software Specification Review  
PDR = Preliminary Design Review  
CDR = Critical Design Review  
TRR = Test Readiness Review  
FCA = Functional Certification Audit  
PCA = Physical Certification Audit  
FRR = Flight Readiness Review  
FQR = Formal Qualification Review

System Definition  
System Development  
System Integration

Daimler Benz Aerospace
## Major Evaluation Data Collection Forms

### Export Form

<table>
<thead>
<tr>
<th>Sender</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipient(s)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date and Time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature of the design work to be sent to the STEP file</td>
<td></td>
</tr>
</tbody>
</table>

|  | 1. Size of design exported (in bytes) |
|  | 2. No. of Data Items? |
|  | 3. No. of Process Elements? |
|  | 4. No. of Design Diagrams? |

<table>
<thead>
<tr>
<th>Other associated documents sent</th>
<th>Document Type? e.g. design document</th>
<th>Media? e.g. paper</th>
<th>Method e.g. post</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>State Any problems encountered during &quot;Export Procedure&quot;?</th>
<th></th>
</tr>
</thead>
</table>
**Import Form.**

<table>
<thead>
<tr>
<th>Sender</th>
<th>Recipient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Date and Time</td>
<td></td>
</tr>
<tr>
<td>Nature of the design work retrieved from the STEP file</td>
<td>1. Size of design imported (in bytes)</td>
</tr>
<tr>
<td></td>
<td>2. No. of Data Items?</td>
</tr>
<tr>
<td></td>
<td>3. No. of Process Elements?</td>
</tr>
<tr>
<td></td>
<td>4. No. of Design Diagrams?</td>
</tr>
<tr>
<td>Other associated documents received?</td>
<td>Document Type? e.g. design document</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>State Any problems encountered during &quot;Import Procedure&quot;</td>
<td></td>
</tr>
</tbody>
</table>

252
**Engineer's Record**

*Author............................
Date...............................*

**Engineer's Record Checklist for export process**

<table>
<thead>
<tr>
<th>Stage of exchange cycle and type of actions to be recorded</th>
<th>Event number(s) of entries in the Engineer's Record Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Preparing Design (in source tool) for Transfer</td>
<td></td>
</tr>
<tr>
<td>• Unexpected snags or problems</td>
<td></td>
</tr>
<tr>
<td>• Interruptions</td>
<td></td>
</tr>
<tr>
<td>• Help sought</td>
<td></td>
</tr>
<tr>
<td>2. Transform of design information from source tool to STEP file</td>
<td></td>
</tr>
<tr>
<td>• Problems with Tool Interface (errors etc.)</td>
<td></td>
</tr>
<tr>
<td>• Data collection actions (obtaining printouts, form filling etc.)</td>
<td></td>
</tr>
<tr>
<td>• Unexpected snags or problems</td>
<td></td>
</tr>
<tr>
<td>• Recovery actions following unexpected problems</td>
<td></td>
</tr>
<tr>
<td>• Interruptions</td>
<td></td>
</tr>
<tr>
<td>• Help sought</td>
<td></td>
</tr>
<tr>
<td>3. Transfer of STEP file via Internet</td>
<td></td>
</tr>
<tr>
<td>• Actions arising through organisational / IT. infrastructure (virus checks etc.)</td>
<td></td>
</tr>
<tr>
<td>• Problems sending file (email failures, Internet delays etc.)</td>
<td></td>
</tr>
<tr>
<td>• Other unexpected snags or problems</td>
<td></td>
</tr>
<tr>
<td>• Recovery actions following unexpected problems</td>
<td></td>
</tr>
<tr>
<td>• Interruptions</td>
<td></td>
</tr>
<tr>
<td>• Help sought</td>
<td></td>
</tr>
</tbody>
</table>
## Engineer's Record Checklist for import process

<table>
<thead>
<tr>
<th>Stage of exchange cycle and type of actions to be recorded</th>
<th>Event number(s) of entries in the Engineer's Record Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Receiving transferred STEP file via Internet</td>
<td></td>
</tr>
<tr>
<td>- Actions arising through organisational / I.T.</td>
<td></td>
</tr>
<tr>
<td>infrastructure (virus checks etc.)</td>
<td></td>
</tr>
<tr>
<td>- Unexpected snags or problems</td>
<td></td>
</tr>
<tr>
<td>- Interruptions</td>
<td></td>
</tr>
<tr>
<td>- Help sought</td>
<td></td>
</tr>
<tr>
<td>2. Transform of design information from STEP file to</td>
<td></td>
</tr>
<tr>
<td>sink tool</td>
<td></td>
</tr>
<tr>
<td>- Problems with Tool Interface (errors etc.)</td>
<td></td>
</tr>
<tr>
<td>- Data collection actions (obtaining printouts,</td>
<td></td>
</tr>
<tr>
<td>form filling etc.)</td>
<td></td>
</tr>
<tr>
<td>- Unexpected snags or problems</td>
<td></td>
</tr>
<tr>
<td>- Recovery actions following unexpected problems</td>
<td></td>
</tr>
<tr>
<td>- Interruptions</td>
<td></td>
</tr>
<tr>
<td>- Help sought</td>
<td></td>
</tr>
<tr>
<td>3. Work carried out (in sink tool) on imported design</td>
<td></td>
</tr>
<tr>
<td>- Problems reading design in sink tool</td>
<td></td>
</tr>
<tr>
<td>- Actions taken to resolve these problems</td>
<td></td>
</tr>
<tr>
<td>- Actions relating to interpretation of imported</td>
<td></td>
</tr>
<tr>
<td>design (rearranging of data flows etc.)</td>
<td></td>
</tr>
<tr>
<td>- Formal queries made to sending engineer</td>
<td></td>
</tr>
<tr>
<td>- Additional material consulted</td>
<td></td>
</tr>
<tr>
<td>- Other unexpected snags or problems</td>
<td></td>
</tr>
<tr>
<td>- Recovery actions following unexpected problems</td>
<td></td>
</tr>
<tr>
<td>- Interruptions</td>
<td></td>
</tr>
<tr>
<td>- Help sought</td>
<td></td>
</tr>
</tbody>
</table>
The complete engineer's record form should contain several copies of this page.

<table>
<thead>
<tr>
<th>Event number</th>
<th>Event</th>
<th>Start Time</th>
<th>End Time</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transform Report Examples

'Concise' Transform Report:

SEDRES Transform Report Format V1.0 11/9/97
Transform direction : Export
Design Tool : Statemate Magnum +
Design Tool version : 1.2.3
Interface version : 2.01
Part21 filename : sedres/strm/export/test4.p21
Date : 12/9/97
Time : 13:42:59
Elapsed time : 00:45

Summary Information

File errors : 0
Syntax errors : 0
Semantic errors : 0
Mapped entity types : 11
Unmapped entity types : 14

Mapped Entity List

<table>
<thead>
<tr>
<th>Tool entity notes</th>
<th>SEDRES entity notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>45 arrow</td>
<td>45 flow</td>
</tr>
<tr>
<td>8 activity occurrence</td>
<td>8 external_agent</td>
</tr>
<tr>
<td>2 chart</td>
<td>1 root_function_item</td>
</tr>
<tr>
<td>4 chart</td>
<td>1 functional_context</td>
</tr>
<tr>
<td>7 data store activity 1,3</td>
<td>7 store</td>
</tr>
<tr>
<td>32 internal activity 1,3</td>
<td>32 function_item</td>
</tr>
</tbody>
</table>

Note
1 Names may be truncated.
3 Substitution of ' ' (space) by '_' (underscore) in names.

Unmapped Entity List

UoF Common Design Principles
   (none)
Requirement UoF
   Critical Issue
   Decision
   Requirement
   Requirement group
System Design UoF
   application
   system
   working mode
Functional design UoF
   (none)
Behavioural design UoF
   Behaviour model
   state
   transition
Data types
   (tbd)
‘Differences’ Transform Report:

SEDRES Transform Report Format V1.0 11/9/97
Transform direction : Export
Design Tool : Statemate Magnum+
Design Tool version : 1.2.3
Interface version : 2.01
Part21 filename : sedres/stm/export/test4.p21
Date : 12/9/97
Time : 13:42:59
Elapsed time : 00:45

Summary Information

File errors : 0
Syntax errors : 0
Semantic errors : 0
Mapped entity types : 11
Unmapped entity types : 14

Mapped Entity List

<table>
<thead>
<tr>
<th>Tool entity</th>
<th>SEDRES entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td></td>
</tr>
<tr>
<td>45 arrow</td>
<td>45 flow</td>
</tr>
<tr>
<td>8 activity occurrence</td>
<td>8 external_agent</td>
</tr>
<tr>
<td>2 chart</td>
<td>1 root_function_item</td>
</tr>
<tr>
<td>4 chart</td>
<td>1 functional_context</td>
</tr>
<tr>
<td>7 data store activity</td>
<td>7 store</td>
</tr>
<tr>
<td>32 internal activity</td>
<td>32 function_item</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Note

1 Names may be truncated.
3 Substitution of ' ' (space) by '_' (underscore) in names.

Unmapped Entity List

UoF Common Design Principles
(none)
Requirement UoF
Critical Issue
Decision
Requirement
Requirement group
System Design UoF
application
system
working mode
Functional design UoF
(none)
Behavioural design UoF
Behaviour model
state
transition
Data types
(tbd)
Detail Entity List

45 arrow
CHG_EQUIP_PARAMS_REQ
NEW_FREQUENCY_REQ
NEW_PRESET_VIA_SPAD_REQ
SCROLL_DOWN_REQ
SEL_NEW_PRESET_REQ
TOGGLE_EQUIP_PARAMS_REQ
TOGGLE_MODE_REQ
TOGGLE_PRESET_REQ

8 activity occurrence
AVIONICS_EQUIPMENT
INPUT_DEVICES
MISSION_DATA_LOADER
DISPLAY

2 chart
CDU_OPERATION
INTERPRET_PILOT_INPUT

4 chart
CDU_OPERATION
INTERPRET_PILOT_INPUT

7 data store activity
EQUIP_STATUS_AND_PRESET_LIST

32 internal activity
UPDATE_EQUIP_OP_PARAMS
INTERPRET_PILOT_INPUT
LOAD_MISSION_DATA
CONTROL_DISPLAYED_INFORMATION

45 flow
CHG_EQUIP_PARAMS_REQ
NEW_FREQUENCY_REQ
NEW_PRESET_VIA_SPAD_REQ
SCROLL_DOWN_REQ
SEL_NEW_PRESET_REQ
TOGGLE_EQUIP_PARAMS_REQ
TOGGLE_MODE_REQ
TOGGLE_PRESET_REQ

8 external_agent
AVIONICS_EQUIPMENT
INPUT_DEVICES
MISSION_DATA_LOADER
DISPLAY

1 root_function_item
CDU_OPERATION

1 functional_context
CDU_OPERATION

7 store
EQUIP_STATUS_AND_PRESET_LIST

32 function_item
UPDATE_EQUIP_OP_PARAMS
INTERPRET_PILOT_INPUT
LOAD_MISSION_DATA
CONTROL_DISPLAYED_INFORMATION

258
Appendix B

Documents related to chapter 4 (The PA study)

THINK-ALOUD TRAINING SESSION .......................................................... 260
PROBLEM STATEMENT ........................................................................... 261
DEBRIEFING SESSION .......................................................................... 262
THINK-ALOUD TRAINING SESSION

Choose one of the following problems and write a program that solves it. You are free to use any notation that suits you e.g. detailed pseudocode, programming language etc. While you are solving the problem, try to verbalise your thoughts as much as you can.

The “Traffic Counting” Problem
"A traffic survey is conducted automatically by placing a detector at the road side, connected by data-links to a computer. Whenever a vehicle passes the detector, it transmits a signal consisting of the number 1. A clock in the detector is started at the beginning of the survey, and at one second intervals thereafter it transmits a signal consisting of the number 2. At the end of the survey, the detector transmits a 0. Each signal is received by the computer as a single number (i.e. it is impossible for two signals to arrive at the same time). Design a program which reads such a set of signals and outputs the following: a) the length of the survey period b) the number of vehicles recorded, c) the length of the longest waiting period without a vehicle."

Fibonacci Sequence Problem
"Compute the first N elements of the Fibonacci sequence. In the Fibonacci sequence, the first two elements are 1, and each element after that is the sum of the previous two elements. For example, given an N of 10, the program produces: 1 1 2 3 5 8 13 21 34 55."

Updating Stock Problem
"Imagine you have a ‘simulated’ computer. The task is to perform transactions as indicated by the transaction file: e.g. '3 13' means 'add 13 to the quantity of item 3 in the old stock file'. By the nature of the 'computer', files can only be traversed sequentially, and so all the transactions for a given item number must be done together. When all transactions for a given item are completed, the value of the old stock cell is sent to the new stock file."
PROBLEM STATEMENT

An N-lift (N-élèveur) system is to be installed in a building with M floors. The lifts and the control mechanism are supplied by a manufacturer. The internal mechanisms of these are assumed (given) in this problem. You and several other designers have been asked to submit proposals for the design of such a system. You may proceed in any fashion you like, but please produce a design that is as detailed as possible without going to the level of code; imagine that your proposal, if accepted, is to be handed over to a junior programmer for implementation.

DESIGN THE LOGIC TO MOVE LIFTS BETWEEN FLOORS IN THE BUILDING ACCORDING TO THE FOLLOWING RULES:

1. Each lift has a set of buttons, 1 button for each floor. These illuminate when pressed and cause the lift to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited (i.e. stopped at) by the lift.

2. Each floor has 2 buttons (except ground and top), one to request an up-lift and one to request a down-lift. These buttons illuminate when pressed. The buttons are cancelled when a lift visits the floor and is either travelling in the desired direction, or visiting the floor with no requests outstanding. In the latter case, if both floor request buttons are illuminated, only 1 should be cancelled. The algorithm used to decide which to service, first should minimise the waiting time for both requests.

3. When a lift has no requests to service, it should remain at its final destination with its doors closed and await further requests (or model a "holding" door).

4. All requests for lifts from floors must be serviced eventually, with all floors given equal priority (can this be proved or demonstrated?)

5. All requests for floors within lifts must be serviced eventually, with floors being serviced sequentially in the direction of travel (can this be proved or demonstrated?)

6. Each lift has an emergency button which, when pressed causes a warning signal to be sent to the site manager. The lift is then deemed "out of service". Each lift has a mechanism to cancel its "out of service" status.
DEBRIEFING SESSION

Clarification of behaviour

• Could you briefly talk me through your writings and comment on the strategies that you followed?

• Did you use any methodologies? If so, which ones?

• (Ask about specific types of behaviour you observed during the session.)

Task complexity

• How difficult did you find the problem?

• Which parts were the trickiest?

• (Provide them with the alternative description of the problem and ask them to choose.)

Experimental method

• Did you find thinking aloud straightforward? How natural was it?

• Do you think it affected your thinking? If yes, in what way?
Appendix C

Documents related to chapter 6 (Experiment #1)

A TUTORIAL IN DESIGN RATIONALE USING QOC ................................................ 264
RANDOMISATION TABLE ................................................................................... 271
EXPERIMENTAL TASKS ..................................................................................... 272
STIMULUS MATERIAL .......................................................................................... 281
QUESTIONNAIRE .................................................................................................. 289
A tutorial in Design Rationale using QOC

Contents

1. Introduction
2. Main QOC elements
3. A simple example: “Shall I take part in the experiment?”
4. The design of the XCL scroll bar
5. Nested rationale
6. A composite example: the “fast ATM” design
7. Exercise
1. Introduction

Design rationale refers to the argumentation that underlies the decisions made during the design of an artefact. There are several notations, techniques and methods for capturing, representing and using design rationale in various design tasks.

QOC is a formalism used to represent design rationale. It is straightforward and expressive - by using simple notions of argumentation you can represent the reasoning behind systems design. Organised in fragments, design rationale can adequately describe argumentation for even complicated systems.

2. Main QOC elements

As its name denotes, QOC’s basic elements are Questions, Options and Criteria. Questions stand for issues that come up during the design task, Options stand for alternative answers to the Question at hand and Criteria are meant to be objectives under which each option is evaluated for fitness. Criteria can be assessed either positively or negatively. Let’s see an example:
3. A simple example: “Shall I take part in the experiment?”

Q: Shall I take part in the experiment?

O: No

C: Prize money

O: Yes

C: Free time
The design of the XCL scrollbar

The Question is: "how wide?" and the Options are narrow and wide. The Criteria are screen compactness and ease of hitting with a mouse. A narrow scroll bar is rejected because ease of hitting with a mouse is more important than screen compactness.
5. Nested rationale

Q: How to display?
- O: Permanent
- C: Low user effort
- C: Screen compactness

Q: How to make it appear?
- O: Appearing
- C: Continuous feedback to user

O: "Natural" cursor movement
- C: Low user effort

O: Scroll button
- C: What the user can do is obvious
6. A composite example: The “fast ATM” design

There is considerable variety in the design of ATMs (Automatic Teller Machines), both in what facilities they provide and in the ways people interact with them. One interesting contrast is between a “standard” ATM (SATM) and a new “fast cash” ATM recently introduced by a British Bank. The SATM offers a range of services, such as balance enquiries, new cheque books or statements, as well as withdrawing cash. The FATM provides only for cash withdrawal. However, more than just restricting services, the procedure for using the FATM is different from the procedure for the SATM. The following figure shows the steps required to get cash from the two ATMs:

The standard ATM (SATM)

1. Push card into slot
2. Type PIN number when prompted
3. Select Cash Withdrawal. (from the several services offered)
4. Select Another Amount (you could have selected one of five preset amounts)
5. Type in amount required and press Enter key
6. Select No (when asked if you would like to request another service).
7. Remove card from slot
8. Take cash from drawer, and receipt from slot

The fast ATM (FATM)

1. Select cash amount (Must be one of six preset amounts)
2. Insert card.
3. Remove card.
4. Type in PIN number.
5. Take cash and receipt from drawer
7. Exercise

The two ATM designs that have just been outlined are compared in the following 3 design rationale fragments (a, b and c). The options referring to the design of the SATM are highlighted using thick black whereas the FATM options are the ones in thick grey. Please fill-in the blank spaces.

a.

The Question is: " ?", and the Options are full range and cash only. The Criteria are variety of services and speed. A cash-only machine would be faster but would restrict the variety of offered services. A full-range machine would provide a larger variety of services but would reduce speed.

b.

Q: Where to retrieve cash and receipt from?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
<th>Different slots</th>
<th>Same slot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
</tbody>
</table>

c.

Q: How to select cash amount?

C: Variety of amounts

C: Obvious what machine provides

C: Speed
Randomisation table

<table>
<thead>
<tr>
<th>Subject</th>
<th>Random number</th>
<th>Within-subjects assignment</th>
<th>Between-subjects assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>17</td>
<td>F (G-&gt;T-&gt;N)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S2</td>
<td>6</td>
<td>B (N-&gt;G-&gt;T)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S3</td>
<td>16</td>
<td>F (G-&gt;T-&gt;N)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S4</td>
<td>4</td>
<td>B (N-&gt;G-&gt;T)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S5</td>
<td>15</td>
<td>E (G-&gt;N-&gt;T)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S6</td>
<td>13</td>
<td>E (G-&gt;N-&gt;T)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S7</td>
<td>2</td>
<td>A (N-&gt;T-&gt;G)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S8</td>
<td>3</td>
<td>A (N-&gt;T-&gt;G)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S9</td>
<td>10</td>
<td>D (T-&gt;G-&gt;N)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S10</td>
<td>1</td>
<td>A (N-&gt;T-&gt;G)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S11</td>
<td>8</td>
<td>C (T-&gt;N-&gt;G)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S12</td>
<td>11</td>
<td>D (T-&gt;G-&gt;N)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S13</td>
<td>14</td>
<td>E (G-&gt;N-&gt;T)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S14</td>
<td>7</td>
<td>C (T-&gt;N-&gt;G)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S15</td>
<td>18</td>
<td>F (G-&gt;T-&gt;N)</td>
<td>B (C-O)</td>
</tr>
<tr>
<td>S16</td>
<td>9</td>
<td>C (T-&gt;N-&gt;G)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S17</td>
<td>5</td>
<td>B (N-&gt;G-&gt;T)</td>
<td>A (O-C)</td>
</tr>
<tr>
<td>S18</td>
<td>12</td>
<td>D (T-&gt;G-&gt;N)</td>
<td>B (C-O)</td>
</tr>
</tbody>
</table>

Results of randomisation procedure
Experimental tasks

TASK 1

Using the given design rationale documents answer the following questions. Write the answers on the space provided on this sheet:

Task 1-A. SYNTAX

1. What is the most promising choice for the range of offered services in terms of speed?

2. What is the FATM option for the initiation of a transaction?

3. Providing the security objective is met, which practical problem is solved?
Task 1-B. SEMANTICS

4. Why there probably shouldn't be a fixed default amount on the card?

5. Does the objective of speed influence the number of slots on the ATM? (Yes/No)
Task 1-C. CONTEXT

6. What is the advantage (if any) of a switchable machine over one with a (permanently) full range of services?

7. Deciding on a customer-programmed card, which cash selection approach would that imply?
TASK 2

*Using the given design rationale documents answer the following questions. Write the answers on the space provided on this sheet.*

**Task 2-A. SYNTAX**

1. What is the most promising choice for the retrieval of cash and receipt in terms of speed?

2. What is the FATM option for selecting cash amount?

3. What is the best way of initiating a transaction in terms of low error rates?
Task 2-B. SEMANTICS

4. As to how to select cash amount, if we chose “both typing and presets”, then what would be our most important criterion in this choice?

5. In the same issue as above (i.e. selection of cash amount), not regarding the importance of individual criteria, what would be our decision and why?
Task 2-C. CONTEXT

6. What is the advantage (if any) of restricting the type of services over a (permanently) full range of services?

7. Deciding on fixed machines, which range of services would that imply?
TASK 3

Using the given design rationale documents answer the following questions. Write the answers on the space provided on this sheet:

Task 3-A. SYNTAX

1. What is the most promising choice for how to restrict the provided services in terms of speed?

2. What is the FATM option for the retrieval of cash and receipt?

3. Would a restriction of the type of services make our ATM transactions slower?
Task 3-B. SEMANTICS

4. In which criterion do we consider intuitiveness in order to provide the right process of cash amount specification?

5. Does the objective of ‘intuitiveness to the user’ influence where to retrieve cash and receipt from?
Task 3-C. CONTEXT

6. What is the advantage (if any) of a customer setting default amount via the ATM over typing in amount straight away (on transaction)?

7. Deciding to restrict the number of services, which range of services approach would that imply?
Stimulus material

The question is: "what range of services offered?". This first option (SATM) is to offer full range and the second option (FATM) is to offer cash only. The first criterion is variety of services and the second criterion is speed.

Offering a full range of services is positive in terms of the variety of services users receive but reduces speed. Offering cash only restricts the variety of services provided but increases speed.

The question is: "where to retrieve cash and receipt from?". This first option (SATM) is to use different slots and the second option (FATM) is to use the same slot. The first criterion is low cost and the second criterion is speed.

Providing different slots for cash and receipt has a low cost but reduces speed. Dispatching cash and receipt through the same slot costs more but increases speed.

The question is: "how to select amount?". This first option is to type in amount, the second option (SATM) is to cater for both typing and presets, the third option (FATM) is to select from presets and the fourth option is to have the amount recorded on card. The first criterion is variety of amounts, the second criterion is "obvious what machine provides" and the third criterion is speed.

Choosing to type in amount provides a variety of amounts to choose from, but it is not obvious what machine provides and it is slow. Having both typing and presets provides a variety of amounts to choose from, but it is not obvious what machine provides and it is slow. Choosing to select cash amount does not provide a variety of amounts to choose from, but it is obvious what machine provides and is fast. Choosing to record amount on card restricts the variety of services, does not make obvious what machine provides and is fast.

The question is: "how to initiate transaction?". This first option (SATM) is to identify customer and the second option (FATM) is to select cash amount. The first criterion is low errors, the second criterion is "obvious what machine provides" and the third criterion is speed.

Choosing to identify customer would give low error rates, but it is not obvious what machine provides and it is slow. Choosing to select cash amount it would be obvious what machine provides and would be faster but would yield higher error rates.
The question is: “does machine hold card?”. This first option (SATM) is to hold card for PIN and the second option (FATM) is to return card immediately. The first criterion is security and the second criterion is speed.

Choosing to hold card for PIN would provide security and reduce speed. Choosing to return card immediately would provide lower security but it would increase speed.

The question is: “how are services restricted?”. This first option is to provide fixed machines and the second option is by switchable machines. The first criterion is speed and the second criterion is variety of services.

Choosing fixed machines would increase speed but restrict the variety of services. Choosing switchable machines would favour variety of services and decrease speed.

The question is: “what is restricted?”. This first option is to restrict the no. of services and the second option is to restrict the type of services. The first criterion is speed and the second criterion is variety of services.

Choosing to restrict the no. of services would increase speed but restrict the variety of services. Choosing to restrict type of services would favour variety of services and decrease speed.

The question is: “how does card record amount?”. This first option is to have a fixed default on card, the second option is that customer sets default via ATM and the third option is that customer programs card directly. The first criterion is speed and the second criterion is variety of amounts.

Choosing fixed default on card is faster but restricts variety of services. Having the customer to set default via ATM, is slower but provides a variety of services. Having the customer to program the card directly increases both speed and variety of services.
The question is: "what range of services offered?". This first option is (SATM) is to offer full range and the second option (FATM) is to offer cash only. The first criterion is variety of services and the second criterion is speed.

Concerning the variety of services offered to users, a full range is preferable than cash only service. Regarding speed, a cash only service is preferable than a full range service.

The question is: "where to retrieve cash and receipt from?". This first option (SATM) is to use different slots and the second option (FATM) is to use the same slot. The first criterion is low cost and the second criterion is speed.

Regarding low cost, providing different slots is preferable than providing the same slot. Concerning speed, a same slot approach is preferable than a different slots one.

The question is: "how to select amount?". This first option is to type in amount, the second option (SATM) is to cater for both typing and presets and the third option (FATM) is to select from presets and the fourth option is to have the amount recorded on card. The first criterion is variety of amounts, the second criterion is "obvious what machine provides" and the third criterion is speed.

Regarding the variety of amounts, typing in amount and both typing and presets are preferable than selecting from presets or recording amount on card. Concerning the obviousness of what machine does, selecting from presets is preferable to typing in amount, both typing and presets, or recording amount on card. Regarding speed, selecting from presets and recording amount on card are preferable than typing in amount or both typing and presets.

The question is: "how to initiate transaction?". This first option (SATM) is to identify customer and the second option (FATM) is to select cash amount. The first criterion is low errors, the second criterion is obvious what machine provides and the third criterion is speed.

Regarding low errors, identifying customer is preferable to selecting cash amount. Concerning "obvious what machine does", selecting cash amount is preferable to identifying customer. Regarding speed, selecting cash amount is preferable to identifying customer.
The question is: "does machine hold card?". This first option (SATM) is to hold card for PIN and the second option (FATM) is to return card immediately. The first criterion is security and the second criterion is speed.

Concerning security, to select card for PIN is preferable than to return card immediately. Regarding speed, to return card immediately is preferable than to hold card for PIN.

The question is: "how are services restricted?". This first option is to provide fixed machines and the second option is by switchable machines. The first criterion is speed and the second criterion is variety of services.

Concerning speed, fixed machines are preferable than switchable machines. Regarding variety of services, switchable machines are preferable to fixed machines.

The question is: "what is restricted?". This first option is to restrict the no. of services and the second option is to restrict the type of services. The first criterion is speed and the second criterion is variety of services.

Concerning speed, restriction of the no. of services is preferable than restriction of the type of services. Regarding variety of services, restriction of the type of services is preferable to restriction of the no. of services.

The question is: "how does card record amount?". This first option is to have a fixed default on card, the second option is that customer sets default via ATM and the third option is that customer programs card directly. The first criterion is speed and the second criterion is variety of amounts.

Regarding speed, "fixed default on card" and "customer programs card directly" are preferable to "customer sets default via ATM". Concerning variety of services "customer sets default via ATM" and "customer programs card directly" are preferable to "fixed default on card".
Q: What range of services offered?
O: Full range
C: Variety of services
O: Cash only
C: Speed

Q: Where to retrieve cash and receipt from?
O: Different slots
C: Low cost
O: Same slot
C: Speed

Q: How to select amount?
O: Type in amount
C: Variety of amounts
O: Both typing and presets
C: Obvious what machine provides
O: Select from presets
C: Speed
O: Record on card

Q: How to initiate transaction?
O: Identify customer
C: Low errors
O: Select cash amount
C: Obvious what machine can do
C: Speed
Q: Does machine hold card?
O: Hold card for PIN
O: Return card immediately
C: Security
C: Speed

Q: How are services restricted?
O: Fixed machines
O: Switchable machines
C: Speed
C: Variety of services

Q: What is restricted?
O: No. of services
O: Type of services
C: Speed
C: Variety of services

Q: How does card record amount?
O: Fixed default on card
O: Customer sets default via ATM
O: Customer programs card directly
C: Speed
C: Variety of services
Q: What range of services offered?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Full range</td>
</tr>
<tr>
<td>Variety of services</td>
<td>+</td>
</tr>
<tr>
<td>Speed</td>
<td>-</td>
</tr>
</tbody>
</table>

Q: Where to receive cash and receipt from?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Different slots</td>
</tr>
<tr>
<td>Low cost</td>
<td>+</td>
</tr>
<tr>
<td>Speed</td>
<td>-</td>
</tr>
</tbody>
</table>

Q: How to select amount?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
<th>Type in amount</th>
<th>Both typing and presets</th>
<th>Select from presets</th>
<th>Record on card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variety of amounts</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Obvious what machine provides</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Speed</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Q: How to initiate transaction?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identify customer</td>
</tr>
<tr>
<td>Low errors</td>
<td>+</td>
</tr>
</tbody>
</table>
### Q: Does machine hold card?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
<th>Hold card for PIN</th>
<th>Return card immediately</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Speed</td>
<td></td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

### Q: How are services restricted?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
<th>Fixed machines</th>
<th>Switchable machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Variety of services</td>
<td></td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

### Q: What is restricted?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
<th>No. of services</th>
<th>Type of services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Variety of services</td>
<td></td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

### Q: How does card record amount?

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Options</th>
<th>Fixed default on card</th>
<th>Customer sets default via ATM</th>
<th>Customer programs card directly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td></td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Variety of services</td>
<td></td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
QUESTIONNAIRE

Please tick (v) as appropriate.

(A) PERSONAL DETAILS

• Sex: M □ F □

• Age: □ □ □ □ □
   15-20 20-25 25-30 30-35 35+

(B) BACKGROUND

• Course: ______________________

• Year of study: □

• Have you taken any Computing or IT-related A-levels? (Yes/No): □

(C) DESIGN PRACTICE

• How many hours per week (approximately) do you think you spend on each of the following or other software design-related activities?

<table>
<thead>
<tr>
<th>Activity</th>
<th>Hours per week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming</td>
<td></td>
</tr>
<tr>
<td>Systems Analysis &amp; Design</td>
<td></td>
</tr>
<tr>
<td>Other (please specify below)</td>
<td></td>
</tr>
</tbody>
</table>

(D) PREFERENCES

289
Please indicate your attitude towards each of the representations you've been exposed to, by marking your response to the following statements on the provided scales. You are encouraged to browse through your responses to the questions throughout the experiment.

**Task 1**

- The notation I used in task 1 is intuitive and easy to comprehend.

  0 1 2 3 4
  |-----------------|-----------------|-----------------|-----------------|
  strongly disagree disagree OK agree strongly agree

- In particular, that is how I would rate its intuitiveness/ease to comprehend over the 3 sub-tasks:

  |-----------------|-----------------|-----------------|-----------------|
  0 1 2 3 4
  Task 1-A (syntax)

  |-----------------|-----------------|-----------------|-----------------|
  Task 1-B (semantics)

  |-----------------|-----------------|-----------------|-----------------|
  Task 1-C (context)

  very low average high very high
• The notation I used in task 1 is suitable for me to understand the design rationale behind the FATM system.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly disagree</td>
<td>disagree</td>
<td>OK</td>
<td>agree</td>
<td>strongly agree</td>
</tr>
</tbody>
</table>

• In particular, that is how I would rate its suitability over the 3 sub-tasks:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1-A (syntax)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1-B (semantics)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1-C (context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

very low low average high very high
- I would be happy to use the notation I used in task 1 to understand design decisions made previously on a regular basis.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly disagree</td>
<td>disagree</td>
<td>OK</td>
<td>agree</td>
<td>strongly agree</td>
</tr>
</tbody>
</table>

- In particular, that is how I would rate its effectiveness over the 3 sub-tasks:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1-A (syntax)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1-B (semantics)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1-C (context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

very low  low  average  high  very high
Task 2

- The notation I used in task 2 is intuitive and easy to comprehend.

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{strongly disagree} & \text{disagree} & \text{OK} & \text{agree} & \text{strongly agree}
\end{array}
\]

- In particular, that is how I would rate its intuitiveness/ease to comprehend over the 3 sub-tasks:

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{Task 2-A (syntax)} & \hline
\text{Task 2-B (semantics)} & \hline
\text{Task 2-C (context)}
\end{array}
\]

\[
\begin{array}{cccccc}
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}
\]
• The notation I used in task 2 is suitable for me to understand the design rationale behind the FATM system.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly disagree</td>
<td>disagree</td>
<td>OK</td>
<td>agree</td>
<td>strongly agree</td>
</tr>
</tbody>
</table>

• In particular, that is how I would rate its suitability over the 3 sub-tasks:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 2-A (syntax)</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Task 2-B (semantics)</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Task 2-C (context)</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

very low  low average high  very high
• I would be happy to use the notation I used in task 2 to understand design decisions made previously on a regular basis.

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{strongly disagree} & \text{disagree} & \text{OK} & \text{agree} & \text{strongly agree}
\end{array}
\]

• In particular, that is how I would rate its effectiveness over the 3 sub-tasks:

\[
\begin{array}{cccccc}
\text{Task 2-A (syntax)} & \hline
0 & 1 & 2 & 3 & 4 \\
\hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}
\]

\[
\begin{array}{cccccc}
\text{Task 2-B (semantics)} & \hline
0 & 1 & 2 & 3 & 4 \\
\hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}
\]

\[
\begin{array}{cccccc}
\text{Task 2-C (context)} & \hline
0 & 1 & 2 & 3 & 4 \\
\hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}
\]
Task 3

- The notation I used in task 3 is intuitive and easy to comprehend.

\[
\begin{array}{ccccc}
   0 & 1 & 2 & 3 & 4 \\
\hline
\text{strongly disagree} & \text{disagree} & \text{OK} & \text{agree} & \text{strongly agree}
\end{array}
\]

- In particular, that is how I would rate its intuitiveness/ease to comprehend over the 3 sub-tasks:

\[
\begin{array}{ccccc}
   0 & 1 & 2 & 3 & 4 \\
\hline
\text{Task 3-A (syntax)} & \hline & \hline & \hline & \hline
\text{Task 3-B (semantics)} & \hline & \hline & \hline & \hline
\text{Task 3-C (context)} & \hline & \hline & \hline & \hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}
\]
The notation I used in task 3 is suitable for me to understand the design rationale behind the FATM system.

\[
\begin{array}{c|c|c|c|c|c}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{strongly disagree} & \text{disagree} & \text{OK} & \text{agree} & \text{strongly agree}
\end{array}
\]

In particular, that is how I would rate its suitability over the 3 sub-tasks:

Task 3-A (syntax) 
\[\begin{array}{c|c|c|c|c|c}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}\]

Task 3-B (semantics) 
\[\begin{array}{c|c|c|c|c|c}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}\]

Task 3-C (context) 
\[\begin{array}{c|c|c|c|c|c}
0 & 1 & 2 & 3 & 4 \\
\hline
\text{very low} & \text{low} & \text{average} & \text{high} & \text{very high}
\end{array}\]

297
• I would be happy to use the notation I used in task 3 to understand design decisions made previously on a regular basis.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>strongly disagree</td>
<td>disagree</td>
<td>OK</td>
<td>agree</td>
<td>strongly agree</td>
</tr>
</tbody>
</table>

• In particular, that is how I would rate its effectiveness over the 3 sub-tasks:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 3-A (syntax)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 3-B (semantics)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 3-C (context)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

very low  low  average  high  very high
If you had to choose one among these three notations for the purpose of comprehending design rationale for computer systems, which one would you prefer? Please tick (✓) as appropriate:

- Narrative
- Tabular
- Graphical

Please provide a brief justification for that decision, in the space below:

- Additional comments:
Thank you very much for your time!
Appendix D

Documents related to chapter 7 (Experiment #2)

DESIGN DOCUMENTATION ................................................................. 302
DESIGN RATIONALE DOCUMENTATION ........................................... 304
Design documentation

Stimulus material – class diagram for the N-lift system
Stimulus material – state transition diagram for the N-lift system
Design rationale documentation

Qa
What type of control?

conditionally implies

Qb
What type of inter-processor communication?

Qc
How to schedule requests?

relates to

Qd
When to process requests?

conditionally implies

Qe
How to organise requests?
Qa: What type of control?
- O: Central
- O: Distributed

C: Ease of synchronisation
C: Reliability
C: No single point of failure

Qb: What type of inter-processor communication?
- O: Sequential baton passing
- O: Broadcast to all
- O: Central store

C: No single point of failure
C: Knowledge of algorithm
C: Ease of algorithm
Qc: How to schedule requests?

O: Closest request

O: Most requests

Qd: When to process requests?

O: On the spot

O: Polling of requests

C: Avoidance of large concentrations of people

C: Minimisation of waiting time for floor requests

C: Retainment of travel direction for lift requests

C: All floor requests must have equal priority

C: Bulk of request

C: Processing speed

C: Easy to implement
Qe: How to organise requests?

- By request status (requesting, servicing)
- By direction (up, down)
- By source (lift, floor)
- By processor (?)
- On FIFO basis

C. Minimisation of waiting time for floor requests

C. Retainment of travel direction for lift requests

C. All floor requests must have equal priority