A paradox of progressive saturation: The changing nature of improvisation over time in systems development projects

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


Additional Information:

- This paper was accepted for publication in the journal Journal of the Association for Information Systems and the definitive published version is available at http://aisel.aisnet.org/jais/vol18/iss11/1

Metadata Record: https://dspace.lboro.ac.uk/2134/25610

Version: Accepted for publication

Publisher: © Association for Information Systems

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.

<table>
<thead>
<tr>
<th>Journal:</th>
<th>Journal of the Association for Information Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript ID</td>
<td>RA-JAIS-14-0137.R4</td>
</tr>
<tr>
<td>mstype:</td>
<td>Research Article</td>
</tr>
<tr>
<td>Keywords:</td>
<td>socio-technical system &lt; Unit of analysis, empirical &lt; Type of research, interpretive &lt; Philosophical Approach</td>
</tr>
</tbody>
</table>
A Paradox of Progressive Saturation: The Changing Nature of Improvisation over Time in a Systems Development Project

Abstract

In this paper, we investigate improvisation in a systems development project within the context of safety-critical, rigid quality management standards. This study took place within a technology company in the automotive industry over a 31-month period and focused on the development of an innovative information system for automobiles. Our analysis traced different forms of improvised practice over the course of a systems development project at the company, along with various triggers of improvisation. We found that as the project progressed, the latitude to improvise became saturated by the increasing structural influences on improvisation. Yet, paradoxically, these structural influences provoked developers to improvise in ways that were progressively more innovative by drawing on accumulated knowledge; we call this phenomenon a 'paradox of progressive saturation'. The study identifies ten forms of improvisation unfolding across different stages of a systems development project. We offer a conceptualization of the paradox of progressive saturation to represent the changing nature of improvisation over time, which contributes to the literature on improvisation in information systems development.

1. Introduction

This paper reports the findings from a case study of systems development to further our understanding of improvisation within the context of safety-critical, rigid quality management standards. In this context, systems developers are challenged by the expectation of having a systematic, planned approach to enable control and predictability despite uncertain and emergent requirements. In addition to these challenges, there is the complexity of having to simultaneously integrate software into emergent hardware innovations; for example, we see this increasingly with Internet of Things (IoT) solutions. To respond to such challenges, it has been established that improvisation is necessary because planned actions alone are inadequate (e.g., Zheng, Venters, & Cornford, 2011). This coping potential of improvisation
has been evidenced before (Magni, Proserpio, Hoegl, & Provera, 2009) and in cognate
safety-critical contexts to ours – emergency and crisis management, for example (e.g.,
Mendonça, 2007). Prior research in information systems (IS) often treats improvisation as
homogeneous throughout a systems development project. For example, improvisation has
been studied within discrete development stages such as design (Teoh, Wickramsinghe, &
Pan, 2012), implementation (Berente & Yoo, 2012), and post-implementation (Rodon, Sese,
& Christiaanse, 2011). Yet, little is known of how the improvisation practices of developers
change across different development stages within the lifetime of one systems development
project. We therefore address two research questions: (i) what kinds of improvisation are
there in a new systems development project within the context of rigid quality management
standards that are characterized by routinized practice? (ii) how do these kinds of
improvisation unfold over the course of the project?

We investigated these research questions through a longitudinal, qualitative field study
conducted in a technology company in the automotive industry that operates in a safety-
critical, ISO-compliant environment. During the field study, we focused on the development
practices and improvisations in relation to an innovative, embedded information system. The
results of the study identify ten major forms of improvised practice at the company. We offer
a conceptualization of the changing nature of improvisation over the course of a single
systems development project.

The rest of the paper is organized as follows: in Section 2, we present our review of prior
research on improvisation in IS, followed in Section 3 by a description of our research
design. In Section 4, we present the results and findings of our empirical study. By drawing
on the themes that emerged from the findings, in Section 5 we discuss our contributions to,
and the implications for, the existing literature.
2. Theoretical Foundations

Firms often use quality management standards such as ISO 9001 in the development of safety-critical software in order to ensure conformity to specific requirements during stages of development such as design, development, production and installation. Firms establish procedures to control and verify design, including planning and design activities, defining organizational and technical interfaces, reviewing and validating design, controlling design changes, and so on. Such process standards are considered as essential for software and systems development that emphasizes rigid quality and safety considerations (Rakitin, 2006; Schrenker, 2006). These standards, broadly based on a document-driven approach to systems development, ensure that the requirements are specified prior to their design and implementation, and that documents are only changed through controlled procedures. In this formal setting, improvisation is regarded as an ad hoc, immature and putatively substandard mode of working (Paulk, Curtis, Chrissis, & Weber, 1993; Bhardwaj et al., 2015). Improvisation is often discouraged by bodies such as the International Standards Organization (ISO) and the Software Engineering Institute (SEI) (e.g., Paulk et al., 1993; Saltz, 2015). They state that if an organization is improvising then they are at the lowest level of process maturity: “In an immature software organization, software processes are generally improvised by practitioners and their management during the course of the project” (Paulk et al., 1993, p.1).

Plans informed by formal standards have been described as inadequate coping mechanisms as challenges and uncertainties arise that disrupt systems development; improvisation has been evidenced to be necessary under these circumstances (e.g., Magni et al., 2009; Zheng et al., 2011). Disruptions take different forms, of course; in putatively highly disruptive contexts such as emergency recovery processes, improvisation is reportedly as important as safety, prediction and planning (Marjanovic & Hallikainen, 2013, pp. 24–32). While there is a move in this literature towards improvisation, it is not necessarily away from formality and planning. Rather, a blended view emerges in which improvisation and planning coexist in
order to deal with highly disruptive situations. Indeed, in the mainstream IS literature, a common interpretation of improvisation is that it involves overlapping forms of IS work, such as planning, designing, and developing (Effah & Abbeyquaye, 2014, p. 12). This has been expressed in terms of dyads in cognate organizational literature on improvisation, such as conception and execution (Moorman & Miner, 1998a), planning and implementation (Moorman & Miner, 1998b), and real-time planning (Miner, Bassoff, & Moorman, 2001).

A systems methodologist will recognize a similar trait in agile methods, which also feature a variety of activities in small, ongoing iterations (Baskerville, Ramesh, Levine, Pries-Heje, & Slaughter, 2003; Karlström & Runeson, 2005). While fluid, these cycles of agile activity are rather more anticipated, whereas with improvisation there is little anticipation of what activity comes next (Lanzara, 1999; Louridas, 1999; Weick, 2001; Stacey & Nandhakumar, 2008, 2009). A further distinguishing feature is that improvisation does not constitute a process model, such as RUP, XP or Scrum (Abrahamsson, Salo, Ronkainen, & Warsta, 2002), but rather is a mixture of capabilities and in situ actions, that is, one does not model improvisation. This overlapping and blending of IS activities has been refined in the so-called ‘paradox’ literature on IS improvisation. For example, proposed paradox pairs include ‘planned serendipity’ and ‘rehearsed spontaneity’ (Mirvis, 1998; Zheng, Venters, & Cornford, 2007). These are paradoxical because it seems inconceivable that a developer could simultaneously rehearse and take spontaneous action. This is an important idea in our paper, and we extend this work by proposing a new paradox pair in our conclusions.

Improvisation is seen as spontaneous in the sense of performing work in an impulse-driven, ‘spur of the moment’ way. It is based on personal inspiration and autonomous reflexivity, where “actors complete their thinking in relative autonomy” (Mutch, 2010, p. 516). Autonomous reflexivity (Archer, 2007) is characterized by an actor’s ability to distance themselves from their working environment and, in so doing, not accept that established custom and practice are the “best ways” (p. 193); they are selective, evaluative, and elective.
They are self-reliant and able to devise courses of action. According to Archer (2007), autonomous reflexivity is particularly applicable when the subject encounters new experiences and novel situations for which their natal context provides no guidelines (p. 194). An autonomous reflexive actor has to learn to rely on their own resources in order to deal with the situation; for example, reflexively generating an innovative solution to overcome a problem created by breakdown of routine or to exploit an opportunity. For instance, an opportunity arose in an IS under development (Njenga & Brown, 2012) that led to the autonomous, spontaneous reconfiguration of the system with new functionality (McGann & Lyytinen, 2008, p. 4). The spontaneous character of improvisation involves the exploration of open possibilities where scripted, routine activity does not apply (Bansler & Havn, 2003). We therefore view improvisation in systems projects as moments of simultaneous planning and execution of actions by developers as they break free or ‘disembed’ themselves from the routine flow of a project with varying degrees of success.

Alternatively, ‘scripts’ may be actively adapted during improvisation, as opposed to being ignored outright, which consequently involves less autonomous reflexivity and rather more anchoring in established practice. This has been shown during new product development (NPD), when improvisation involved mixing and matching established procedures in novel ways (Moorman & Miner, 1998b, p. 703; Pavlou & El Savy, 2010). Indeed, successful improvisations have been shown to draw on accumulated knowledge (knowledgeability) and experience per se (Cunha, Cunha, & Kamoche, 1999; Dybå, 2000; Zheng et al., 2011, p. 6; Weick, 1999; Bansler & Havn, 2003). Improvisation involves the developer in an unwitting entanglement with established organizational customs (Archer, 2007) that are part of their repertoire (Zheng et al., 2011, p. 6).

The sway between autonomous reflexivity and established practice yields different levels of improvisation, which generally range from radical improvisational creativity to modest shifts in behavior (Weick, 1993). Moorman and Miner (1998b, p. 703) classify three degrees of
improvisation: slight adjustments to preexisting processes, stronger departure from existing practices, and the most extreme level, where “the improviser discards clear links to the original referent and composes new patterns”. Hence, in its extreme form, improvisation connotes “producing something on the spur of the moment” (Weick, 1998, p. 544). This links back to the idea of spontaneity and autonomous reflexivity. Weick (1998) uses the metaphor of jazz improvisation (cf. Gioia, 1988, p. 66) as a way of illustrating the various “degrees of organizational improvisation”. For example, he argues that pure instances of improvisation are activities that alter, revise, create, and discover, that are imaginative and creative; on the other hand, activities that shift, switch, or add are at the less imaginative end of the continuum. Weick (1998, p. 554) claims that jazz improvisation “teaches us that there is life beyond routines, formalization, and success”. Activities happen outside organized routines or formal plans (Miner et al., 2001).

Our literature review has also revealed that there are limited insights into how improvisation is accommodated within a systems setting characterized by formal quality standards. IS literature often treats improvisation in systems development projects as singular or homogeneous within the lifetime of a systems project. We extend existing research by studying the nature of improvisation occurrences across multiple development periods within a single safety-critical project. We draw on Archer’s (2007) concept of autonomous reflexivity in order to flesh out an aspect of improvisation that is often pointed to but not unpacked to any significant extent, that is, personal inspiration (Bansler & Havn, 2003; McGann & Lyytinen, 2008, p. 4), which is also an important concept for our case analysis.

3. Research Design and Empirical Context

In this section, we describe: (i) our research approach; (ii) site selection and the timeline of the research project; (iii) the case study company ‘ImproCo’; (iv) the data collected; (v) how the data was analyzed.
3.1 Research Approach

Our study investigates a systems development process in its context, and constructs an understanding of the participants’ activities in that particular setting over a period of time. As such, a longitudinal, qualitative field study (Walsham, 1993) was conducted at ImproCo (a pseudonym), which is a high-tech company operating in Germany's automotive industry. This involved the collection of detailed, qualitative data through a combination of interviewing key actors involved in a new technology development project in their natural setting, and carrying out observations of the practices, and reviewing documentation, over a period of 31 months at ImproCo.

3.2 Site Selection and Timeline

Site selection was influenced by “theoretical relevance and purpose” (Orlikowski, 1993, p. 312). The main focus was investigation of the challenges of the innovative development of new technologies within a ‘rigid’ context of quality management standards characterized by routinized practices. In order to accomplish this aim, we selected ImproCo, because it was an ISO 9001-certified organization with the relevant contextual elements. For example, senior management implemented the core ISO 9001 processes of business acquisition, design and development, test, production, and delivery, service and support, as well as the supporting processes of business management, supplier management, inventory management, and configuration management. These were documented in the organization's quality manual, which guided and served to integrate ImproCo's development processes. Adhering to quality management standards was regarded as necessary by senior management in helping to manage organizations through structured processes.

3.3 The Case Study Company

ImproCo provides high-tech systems and services for the research and development centers of automobile manufacturers in Germany. These services and products involved solutions for their customers that ImproCo described as “innovative and tailor-made”; their business success led to an expansion of the company – at the time of writing, ImproCo employed 180
people. Because of the growth of ImproCo, more alignment of their processes was required and ImproCo managers introduced a quality management standard that was compliant with ISO 9001. ImproCo’s quality management standard covers generic processes for the entire organization, as well as specific ones, including administrative procedures and evaluations of employees and customers. ImproCo’s senior management team expected the ISO 9001-certified procedures to be followed (clause 7.3 of ISO 9001, that is, that design and development should have distinct, linear stages). Hence, ImproCo’s developers principally followed a linear lifecycle process, which proceeded as follows: requirements definition, concept development, initial proof of concept, and, in the final stage, - testing and quality checks.

3.3.1 Project Description

Our study at ImproCo focused on exploring the improvisational practices that were carried out during the development of an in-car information system, within a quality management context. The in-car information system, built on Linux, was designed to help developers collect and analyze data transmissions, that is, between the various electronic units within a car, such as the central unit and the CD changer, during automobile development. This involved three different, interconnected systems and activities: hardware development, embedded software development, and client software development (data download). These different activities involved various internal stakeholders (software developers, hardware developers, development managers, project managers, quality managers, and business managers for marketing and customer relations) and external stakeholders (one principal organizational customer with multiple divisions), and necessitated complex discussions and negotiations.

3.4. Data Collection at ImproCo

We interviewed and observed several actors that were involved in the systems development project that we studied at ImproCo (Table 1). This was part of a larger research project that aimed for a better understanding of improvisation during a new technology development
project. One of the actors, Scott (a pseudonym, as with all the names used in this study), occupied a role of critical importance in the project we investigated. The company’s employees portrayed him as a “fire-fighter”, someone who helped them whenever they were stuck on a problem. Scott described himself as “a self-taught person with experience and passion”. Another actor, Robert, was Scott’s supervisor – he was responsible for the development department that consisted of 15 people involved in the design of hardware and software for this system. Robert engineered the system architecture, which strategically influenced the ongoing development of the in-car IS device. Another actor, Jack, was the project manager and his role was vital to the success of the project because he was also well acquainted with the customer.

### Table 1: Data collection at ImproCo

<table>
<thead>
<tr>
<th>Observations</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>248 visits</td>
<td>Visits to ImproCo over a period of 31 months</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Interviews</th>
<th>Type of interview</th>
<th>Name (pseudonym)</th>
<th>Responsibility of interviewees</th>
<th>Years of experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Group</td>
<td>Group</td>
<td>Project under study</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Semi-structured</td>
<td>Other developers</td>
<td>Software &amp; hardware development</td>
<td>Ranged between 5–15 years</td>
</tr>
<tr>
<td>2</td>
<td>Semi-structured</td>
<td>Scott</td>
<td>Software &amp; hardware developer, “fire-fighter”</td>
<td>&gt; 15 years</td>
</tr>
<tr>
<td>3</td>
<td>Semi-structured</td>
<td>Robert</td>
<td>Development department manager</td>
<td>&gt; 15 years</td>
</tr>
<tr>
<td>3</td>
<td>Semi-structured</td>
<td>Jack</td>
<td>Manager of the studied project</td>
<td>&gt; 10 years</td>
</tr>
<tr>
<td>2</td>
<td>Semi-structured</td>
<td>Mike</td>
<td>Software developer, graphical user interface specialist</td>
<td>&gt; 5 years</td>
</tr>
</tbody>
</table>

Total 20

A timeline of the project, including data collection activities, is depicted in Figure 1. We conducted 20 interviews (2 group interviews and 18 semi-structured interviews), each lasting approximately 90 minutes and conducted at the research site. While ImproCo had more than 180 staff members, we focused on a team of developers and managers that participated in one development project. The lead researcher commenced the field study with two group
interviews, involving all developers and managers, in order to introduce both themselves and the research project. In addition, in accordance with a non-disclosure agreement, the researcher assured participants of their anonymity. After these group interviews, 18 semi-structured interviews were conducted by the lead researcher, in which open-ended questions were asked in order to guide the interview and promote the opportunity to collect rich data through more extensive responses. Interviews were carried out in two stages: those in the early part of the data collection mainly focused on individuals’ perspectives of the development process, emerging issues and the key socio-technical challenges in the project context; towards the end of the data collection, the purpose was mainly to clarify project participants’ perspectives of various observations made (see Figure 1). To support the research process, the researcher tape-recorded the interviews (about 26 hours) and transcribed and translated them from German into English (more than 350 pages of transcription and notes).

Figure 1: Timeline of the project with data collection efforts

In addition, between December 2004 and June 2007, the lead researcher spent two normal working days a week at the research site with the development team; this was to observe their environment and practices and entailed 248 visits. Each visit involved mainly non-participant observation (Leidner & Jarvenpaa, 1993; Nandhakumar & Jones, 1997), where the researcher took notes (more than 150 pages in total) of various events and activities of people, including: (i) formal and informal meetings; (ii) conversations between developers and/or managers; (iii) behavior during the development activities (mainly related to the
project under study). Although we were not able to observe everything that happened, as the research progressed we identified and focused on key people and events, and able to uncover the team’s routines in their everyday work setting. This enabled the lead researcher to gain personal experience of the research context under normal conditions and to get behind the official picture (Goffman, 1959). Further, company documents were analyzed to obtain better insights into the context and processes of ImproCo’s development activities and to support the review of interview and observation notes.

3.5 Data Analysis

The field notes from observations, interview transcripts, and company documents were read and re-read to become acquainted with the data and distinguish meaningful events and incidents. With the amassed field notes, the researchers were able to retell the story of ImproCo’s development activities in detail (Dyer & Wilkins, 1991).

Broadly speaking, the data analysis process involved three interconnected steps. First, as the data collection progressed, we performed descriptive coding (Miles & Huberman, 1994) of the interview transcripts, observation notes, and other material, such as project and training documentation. In order to address the key research aim, the initial coding focused on identifying and highlighting extracts that described team members’ improvisational activities and the changes to their practices throughout the project. We also analyzed the context of these activities and practices, such as associated emergent social and technological constraints pertaining to hardware and software, and tensions around following ISO procedures and standards.

Second, by examining these codified extracts from interviews and field notes as the analysis progressed, we identified early patterns of instances of different improvisational practices in their context. This resulted in us identifying clusters of improvisational practices (first-order categories) from the data and their related contextual aspects, such as socio-technical
issues, over time. In order to understand these emerging patterns in the data, we traced the occurrence during the project of each form of improvisation in our analysis of field data by zooming-in (Nicolini, 2009) on developers’ day-to-day activities, noting incidents of improvisation and when they ceased to occur.

Third, having identified various forms of improvisation, we further analyzed them in order to develop higher-order categories (second-order themes), which reflected the tensions and ‘anchors’ relating to ongoing practices or developers’ spontaneous actions over the course of the project. This process was iterative and nonlinear. Table 2 provides an example of how the analysis process unfolded, with a more detailed version available in Appendix 1.

<table>
<thead>
<tr>
<th>Table 2: Stages of Data Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stage</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

### 4. Empirical Findings

As outlined above, the study focused on the unfolding of the improvisational practices at ImproCo over the course of the development of an in-car information system. We present the process as having three development ‘periods’ (Langley, 1999, p. 703), which are based on our observation of three major tensions as the process unfolded. Thus, Period 1 involved
following formal plans of action while coping with emerging requirements; the concreteness of the contractual planning activities between ImproCo and its client contrasted with radically changing requirements as prototyping was carried out. Period 2 involved allowing the flexibility to change while stabilizing the system; having established a proof of concept during Period 1, the designs were more concretely implemented during Period 2, thereby stabilizing the system, yet client requests for changes to the system were unrelenting. Period 3 involved the developers finding imaginative workarounds while enforcing quality standards; as the project came to a close, with various sign-offs looming, developers' workarounds to problems had to be ever more imaginative. Each of these periods had a cyclical element, in terms of ongoing testing, debugging, and review. We depict the timeline and key events per period in Figure 2, which is not meant to be exhaustive but highlights some key events.

Figure 2: Timeline of development process

While the quality management context provided strict guidelines and plans for the evolving process, throughout the project there were continuous tensions between following a prearranged plan and adjusting to the need of the moment. These tensions constituted a formative context for a variety of triggers and forms of improvisation. We elaborate on these tensions as 'roots' of improvisation on a period-by-period basis in the following sections.
4.1. Following formal plans of action while coping with emerging requirements

*(Period 1)*

The starting point of the development process was a request from a customer to ImproCo, which initiated the development of a custom system for that customer. Each request was treated with great enthusiasm, as the developers were eager to work on novel and innovative products. Development activities within the automotive industry tend to be highly methodological, so developers mostly follow some sort of routine in their practices, consistent with quality management procedures. Some of the formal elements were: (i) formal meetings to discuss budgetary, time and human resource constraints; (ii) client sign-off of agreed milestones, costs to the client, and human resource commitments on the part of the systems developer; (iii) implementation and circulation of these agreements to internal management and development teams; (iv) assignment of work to developers and the estimation of dependencies; (v) readying of the systems management environment, such as bug-tracking software to record and manage various system bugs accordingly. Formal elements (i)–(iii) correspond to the core ISO 9001 process of business acquisition, and elements (iv) and (v) correspond to the core ISO 9001 process of design and development. However, not all aspects of development activities were foreseeable and the developers had to ‘tinker’ with the system. Such tinkering practices were not officially encouraged at ISO-compliant ImproCo, and as the development process progressed it became increasingly hard to accommodate such practices.

The initial period of the project was marked by many unknowns and the developers had to respond to fluctuating requirements. They were still defining the proof of concept and ‘sketching’ the embedded system architecture. At this stage, the freshly formed project team found it challenging to make sense of the customer’s vague design requirements while following the formal plans and development procedures for the project. For instance, there were no clear requirements for any connectivity ports for the all-in-one device under development. As Robert recalled:
The customer wanted to have a USB [Universal Serial Bus] connector at the beginning, and when we stated that this would influence the electromagnetic characteristics of the sensible environment within an automobile, the customer decided to skip this feature. Therefore, we continued further development without a USB ... [but later on] the customer wanted to have this feature again, although the system's architecture provided no availability for a USB connection in terms of hardware or software driver.

Such fluctuating requirements often forced the project team to change plans, to be more interactive with the customer, and to think ahead. Hence, the developers were continuously engaged in modifying plans while executing them.

Individual developers perceived their key role as providing the best solution for the client, even at the cost of breaking some design rules. This self-perceived need to deliver ‘proper’ solutions led to constant ‘tinkering’ and revision of the software code before the specifications became stabilized. This helped to address the emerging issues and technical challenges within the customer's contexts. For example, the developers, of their own choice, revised software code as they became more knowledgeable about the customer’s needs and what was needed to get the system working. They converted the initial concept into a formal list of requirements for the three different development areas of hardware, embedded software, and client software. Although the developers had extensive experience of developing the hardware and embedded software aspects, they were unfamiliar with developing client software. Robert recalled:

\[ I \text{ did not expect to extend the client software towards an application with a graphical interface. I thought a line-based console application would just do the job! } \]

Jack elaborated on the uncertainty:

\[ \text{For me, there was no doubt that we would need a graphical interface because the users of this electronic device would like to have a simple solution which does not require additional training. However, the initial ideas did not involve any user-friendly approach, so we needed to adapt to that principal requirement.} \]

Mike spontaneously developed code for this part of the software, while tinkering with the latest models of the physical device, consisting of hardware and embedded software. However, the development of these three main system aspects was interdependent and
iterative. Irrespective of the original project plan, we observed such voluntary tinkering in the
development of the software throughout the project. However, given the strict quality control
regime, the developers were not keen to disclose such contradictory practices to
management; they instead gave the impression that software releases for the customer
followed the quality procedures. Nevertheless, behind the scenes, a great deal of self-
motivation and inspiration triggered different forms of improvisational practice. For example,
Scott lost his sense of time during a weekend as he spontaneously and freely examined
samples of the hardware, trying to reveal the root cause of the recurrent errors. This involved
one particular problem that had perplexed the developers for some time – were the errors
being ‘thrown’ by the embedded software or the hardware? Scott claimed that some of the
printed circuit board (PCB) tracks were poorly designed, exhibiting mismatched processor
clock-time, and that the ‘paths’ of some integrated circuit pins were too long, all of which
caused delays in the signals. In a developers’ meeting, they decided to keep that PCB and
experiment by soldering some tracks of the hardware. This enabled the developers to
continue with their routine work on the development of the embedded software. In addition,
they decided to find another supplier for the next hardware samples, in order to try to
eliminate quality problems. Jack and Mike were unaffected by these problems, however, and
they continued to design a new graphical interface. This interface was not part of the initial
idea nor was it a customer request, but Jack and Mike saw it as important and developed it
from scratch as an innovative solution. While they attempted to develop the interface
carefully, they admitted to applying some ‘shortcuts’ for testing purposes. However, they said
that these shortcuts did not become part of the official releases. During various coffee
breaks, the team discussed Scott’s ‘bold move’ in isolating the PCB-related error – they were
relieved that the error was not caused by them but by the supplier. This was important to
them because they did not want the management to doubt their technical ability to produce
high-quality systems. Scott spent the rest of the day in discussions with the hardware
developers, to further investigate the errors and try to identify a tactic to mitigate them in any
new hardware samples provided.
Although ImproCo’s ISO 9001-certified procedures and tactics prescriptively shaped their practices, the designers found them less helpful for dealing with the day-to-day design challenges that emerged in the project. The designers sought to appropriate known practices differently to cope with difficulties during the development project, such as frequently changing requirements. For example, technical difficulties arose when the developers created a key feature of the embedded system (although this would have worked fine in a proof-of-concept environment following known practices). Subsequently, they had to twist the known practices to exploit the emerging capabilities. For example, during the project the entire hardware platform presented an unexpected challenge as ImproCo moved from a 16-bit single-processor platform to a 32-bit multiprocessor platform. The developers had to develop new software code to deal with the technological leap, which created some challenges during the development because the complexity of the hardware, the software, and its interaction all grew. This was not captured in the original agreements, and meetings had to be held and paperwork raised to flex the budgets as a result.

As the project progressed, other new constraints emerged, resulting from design features based on the incomplete specification. The draft specification was written shortly after the initial idea, and key parts of the requirement were ill-defined and incomplete. Therefore, continuous adjustment of development activities became part of the work during the initial period of the project. Developers increasingly became more pragmatic and tried out new things, and often discovered new ways of resolving problems and appended them to the project. Many such discoveries needed to remain somehow within the project scope, because of time constraints and the need to realize the project goal. One of the developers commented on this balancing act:

…we had to follow a moving target. Unclear descriptions of the requirements obstructed our development efforts. Those descriptions were unclear, because the customer did not really know what he wanted. So he came out with the ‘wise’ solution to integrate every possible idea. It was our task to put the features into a state which was realizable. So, we had to investigate and discover features to integrate and features to omit.
As activities became more structured over time and the design became more stable, the developers found it harder to actualize their new ‘discoveries’.

4.2. Allowing the flexibility to change while stabilizing the system (Period 2)

The next tension we identified in the data relates to ‘seeking flexibility to change within a stabilizing system’. As the project team began to make good progress in resolving much of the ambiguity within the requirements and stabilizing the product specification, they also faced increasing requests for change from the customer as they moved into the middle period of the development. Hence, during this period the designers tried to be flexible, seeking to alter hardware and software features in response to the customer’s ongoing requests and related shifts in circumstances. While trying to stabilize the data storage aspects of the system, customer ambivalence necessitated developer flexibility, as Jack explained:

Another challenge was the undecidedness of the customer, which resulted in the problem of a moving target. For example, besides the issue with the USB connector, we initially planned to include a Compact Flash connector for the device, so that users might have been able to extend the data storage capability by flash memory. Although the project stakeholders agreed on the reasonableness of the extension bay, we needed to skip the Compact Flash connector at an advanced stage of the development because the customer suddenly decided to leave it out.

A fortunate consequence of the episode described by Jack was that the hardware architecture was stabilized. However, the software architecture still needed to be tailored to the exact technical setting of the various chips and processors of the system. In this context, the tailoring of the embedded software architecture required frequent tinkering and learning-by-doing until the developers (and their managers) were sufficiently confident of their solution.
In order to handle the increasing numbers of customer requests, and as the products became more complex and challenging, the managers and developers decided to change design practices in an effort to adapt to the new situation. For example, Jack recalled:

_This project was not planned to develop a system for mass production. The prototype development involved only a handful of devices. After we had gotten the supplier’s contract, we talked about 80 or 90 devices. However, we are now talking about 800 devices and we originally thought that 80 or 90 devices would be the maximum. So, that’s a huge difference, not only in terms of development, marketing, and production, but also in terms of maintenance. Indeed, over time we were able to adapt our development approach to this increasing number._

The team had to scale up and adapt their practices as they dealt with the emerging situation. Although their ability to prepare and plan for all eventualities was limited, managers and developers were able to channel their knowledge in response to the emerging challenges. For instance, the hardware element was becoming more stable and they were aware of the potential side effects of increasing the system’s complexity. In view of this, they had to respond quickly to new demands and challenges in order to keep the project on schedule. A good example of this was how the graphical interface team dealt with incoming customer requests that disrupted their workflow. Customers asked for advice on such things as working with the software, incidents with the software, and solutions to bugs. Fortunately, the project team maintained a bug-tracking system, so that arising complexities such as incidents, requests and bugs were recorded and managed accordingly. Another example of responding quickly to new demands and challenges surrounded the PCB problem is described above. Continuous interruptions to deal with necessary administrative work and discussions with hardware suppliers, other developers, and the project manager all thwarted Scott’s attempts to resolve it. Although he managed to devise an innovative workaround, he explained to the team that it was only a temporary solution, which could potentially introduce further problems.
4.3. **Finding imaginative workarounds while enforcing quality standards**

*(Period 3)*

The next tension we identified in the data relates to imaginative and innovative workarounds within an ISO-oriented culture. As the new system was nearing completion, with various sign-offs looming, the customer became even more engaged and was making several last-minute requests for change. Interactions with the customer became very tense as existing practices for responding to such requests became disruptive to the work context. Therefore, the team rapidly devised imaginative ways of responding to urgent requests for system changes. Jack explained:

*On a whim, I invited the representative stakeholders to discuss urgent project issues. The growing complexity required this meeting in order to find best approaches for further progressing [the project]. In addition, we required another approach to stay on schedule. I realized that the way we communicated during the project was getting unmanageable. In order to get things done more properly, I started to invite the project stakeholders for meetings, so that we were able to discuss the immediate issues and decide on how to proceed. Those issues involved just another change in the requirements and the discussion of additional system features. However, not all issues were resolved and, over time, it became a standard procedure in the form of a weekly meeting, which now is part of our work setting."

This was imaginative in the following sense: at a point in the process when the project was becoming ever more structured and with ISO-based sign-offs looming, one would not expect such a radical meeting to take place. Yet Jack realized the need for this "on a whim", as he put it. The tense interaction with stakeholders also inspired the team to experiment with different ways to organize activities such as customer support. For example, Jack tried out different modes of interaction with customers, and rearranged responsibilities with the aim of making this process better. This inspired new ways of handling interactions with stakeholders and freed up developers’ time to deal with the design work more efficiently. However, their latitude to find effective workarounds was increasingly constrained by the growing stabilization of the systems. For example, until this point in the project the individual developers had been testing their own code for the various system components for which they were responsible, such as the Controller Area Network (CAN) chip, digital signal processing (DSP) chip, embedded Linux software and client software. This approach to
testing had been successful in previous projects for the company, so it seemed normal to
them, but the current project was becoming increasingly complex. In order to coordinate
testing practices and raise the quality of the product, the senior developers decided to make
team testing more regimented by introducing system integration testing.

As outlined above, despite all their efforts to resolve matters 'on the hoof', it was not always
successful; malfunctions sometimes occurred for no apparent reason, although the
increasing complexity of the system may have played a part. As a result, individual “star
developers” had to come to the rescue by creating solutions spontaneously.

Developers and managers considered Scott an important team member because he was
able to deal with project challenges spontaneously. Although most developers had their
‘eureka’ moments, the developers and managers were particularly impressed by Scott’s effort
in doing this on a frequent basis, thereby solving hardware and software problems
throughout the project, but particularly during this critical, latter stage of the project when
resource budgets were running out. Scott admitted, however, that his skills were not confined
to know-how from the hardware and embedded software domains, but also reflected an
intense desire and dedication to finding the cause of a problem. He described his approach
not as “tinkering” but as “goal-oriented tasking; somewhere between creative chaos and
structure”.

4.4 Summary

The findings above, pertaining to three periods of the development project, highlight the key
forms of improvisation in a context consisting of a variety of socio-technical challenges, such
as rigid quality management standards, ongoing shifts in customer circumstances, routinized
practices, and hardware and software constraints. Table 3 summarizes these findings in
terms of the associated forms of improvisation, and their triggers, contexts, and anchors.
Table 3 shows the 10 forms of improvisation that we were able to induce, and which arose from tensions in the development process and context, together with case examples of triggers for each form of improvisation (column 2). In the fourth column, we introduce ‘Anchors of improvisation’ which are conceptual descriptors for each improvisational form; drawing on concepts from the literature review, they reflect whether each respective improvisational form was based on autonomous reflexivity, ongoing practices, or a mixture of both. We discuss this table further in the next section.
### Table 3: Triggers of improvisation at ImproCo

<table>
<thead>
<tr>
<th># Form of improvisation</th>
<th>Case example of triggers</th>
<th>Socio-technical context</th>
<th>Anchors of improvisation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tension: Following plans of action/Emerging requirements (Period 1)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Planning while executing</td>
<td>Scheduled work interrupted by fluctuating customer requests during the USB episode.</td>
<td>Fluctuating requirements creating a 'second guess' environment of features customers might need; e.g., USB port which interrupted scheduled work for a time.</td>
<td>Mix of autonomous reflexivity and ongoing practices</td>
</tr>
<tr>
<td>2. Revising voluntarily</td>
<td>Developers' self-perceived need to deliver proper solutions for the customer; e.g., the anticipation that a graphical interface would be preferred by the customer for the software 'client'.</td>
<td>Developers became more knowledgeable of customer needs while the hardware and software were still malleable.</td>
<td>Based on personal inspiration and autonomous reflexivity</td>
</tr>
<tr>
<td>3. Appropriating known practices differently (remixing)</td>
<td>ISO 9001 tactics were less helpful and the developers had to appropriate practices differently, using common sense, as the complexity of the hardware/software grew.</td>
<td>Strict ISO regime and challenging 32-bit hardware platform.</td>
<td>Based on ongoing practices</td>
</tr>
<tr>
<td>4. Discovering and appending</td>
<td>Unknown/unclear initial requirements – developers decided to integrate every possible idea to discover features, then decide whether to integrate or omit them.</td>
<td>Pressure to realize project goal while the hardware and software still afforded development of new features.</td>
<td>Based more on personal inspiration and autonomous reflexivity</td>
</tr>
<tr>
<td><strong>Tension: Allowing flexibility to change/Stabilizing system (Period 2)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Altering design features in response to shifting circumstances</td>
<td>Although the customer originally wanted the feature that extends the data storage capability by flash memory, they later decided they did not want it.</td>
<td>Indecisive customer and hardware challenges.</td>
<td>Based on a mix of autonomous reflexivity and ongoing practices</td>
</tr>
<tr>
<td>6. Changing design practices and adapting to new situations</td>
<td>The volume of the devices demanded by the customer was massively revised – the team had to scale up their design practices to cope.</td>
<td>Shifting customer needs/development circumstances and complexity of the hardware and software.</td>
<td>Based on a mix of autonomous reflexivity and ongoing practices</td>
</tr>
<tr>
<td>7. Channeling knowledge to respond to emerging challenges</td>
<td>Refined requirements and understanding but growing complexity.</td>
<td>Acquiring practical insights about the contextual issues of customer environment and better sense-making of technical aspects.</td>
<td>Based more on ongoing practices</td>
</tr>
<tr>
<td><strong>Tension: Finding imaginative workarounds/Enforcing quality standards (Period 3)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Devising responses to urgent needs</td>
<td>Communication overload with customers due to their increased demands. Weekly meeting routine established.</td>
<td>Tense interaction with stakeholders while the system was becoming more stable.</td>
<td>Based more on ongoing practices</td>
</tr>
<tr>
<td>9. Experimenting with work organization</td>
<td>Having set up a weekly meeting routine with customers, this inspired a new approach to coping with customer support too. Concurrently, system integration testing also introduced a more structured approach.</td>
<td>Tense interactions and structured work organization/expectation.</td>
<td>Based more on ongoing practices</td>
</tr>
<tr>
<td>10. Creating solutions spontaneously</td>
<td>Responding immediately to malfunctions – e.g., the 'paths' of some connected pins from different integrated circuits were too long, which meant the signal transmission was delayed and caused malfunctions.</td>
<td>Management style supporting individual problem solving and approach to time, as well as novel technology experimentation.</td>
<td>Based on personal inspiration and autonomous reflexivity</td>
</tr>
</tbody>
</table>
As summarized in Table 3, each of the 10 forms of improvisational response was anchored in autonomous reflexivity or ongoing practices or a combination of the two. These forms of improvisation were also broadly associated with specific periods of the project. By combining the anchors and periods in Figure 3, we seek to depict the unfolding patterns of the improvisational forms at ImproCo. Figure 3 should be considered more as a ‘floor sketch’ of movements over the project’s duration than as a graph. Hence, the ‘downward’ slope of the lines indicates improvisational forms changing their anchors over time, moving from autonomous reflexivity and becoming more related to ongoing practices. The improvisational forms at the top of the figure are more anchored in developers’ autonomous reflexivity. We discuss Figure 3 in this section according to the three periods and three associated anchors.
As shown in Figure 3, during Period 1 of the project the improvisational forms ‘Revising voluntarily’ (#2) and ‘Creating solutions spontaneously’ (#10) were anchored to autonomous reflexivity. The former happened spontaneously during the design of the new graphical interface and other improvements, and the latter involved the disregard of organizational practices. Thus, the triggers for autonomously reflexive improvisation were self-motivation and inspiration.

While the improvisational form ‘Discovering and appending’ (Figure 3, #4) occurred in Period 1, it became progressively more anchored in ongoing practices in Period 2, becoming a
mixture of autonomous reflexivity and ongoing practice. ‘Planning while executing’ (Figure 3, #1) and ‘Changing design practices and adapting to new situations’ (Figure 3, #6) occurred in Periods 1 and 2, becoming progressively less frequent with structural constraints. However, developers were still able to disregard some of these constraints and were able to devise innovative solutions, as illustrated by Scott’s temporary fix to the persistent PCB problem. Therefore, there were improvisational forms in the early and middle period of the project that were triggered initially by self-motivation and inspiration, but as the project progressed the triggers became more related to context.

Transitioning to the third and final period of the project, the improvisational forms ‘Altering design features in response to shifting circumstances’ (Figure 3, #5) and ‘Channeling knowledge to respond to emergent challenges’ (Figure 3, #7) became progressively more anchored more in ongoing practices. Despite the increasing degree of structure, ‘Channeling knowledge to respond to emergent challenges’ (#7) continued longer as the new product accommodated some of the changes. As per Figure 3, the improvisational form ‘Devising responses to urgent needs’ (#8) became more observable in these final stages of the project, becoming more anchored in ongoing practices. Other forms, including ‘Experimenting with work organization’ (Figure 3, #9) and ‘Appropriating known practices differently’ (Figure 3, #3), were also anchored more in established practice, the latter being particularly ISO-based. These established practices involved the procedures of quality management standards in the automobile industry as well as the routinized practices of the systems development project team. While these ISO-based practices and structures were present in the early and middle periods too, the developers were able to improvise around them. The triggers for the forms of improvisation associated with this later stage were more related to socio-technical context, with developer efforts to find effective workarounds increasingly shaped by growing resistance of a social and technological nature; for instance, the introduction of system integration testing in Period 3 of the project.
Table 4 takes the discussion of Figure 3 forward by deriving three ‘levels’ of improvisation for the forms we discovered in the data: fluid, mélange, and anchored; in turn, these are based on autonomous reflexivity (top of Table 4), the anchor of ongoing practice (bottom of Table 4), or a mixture (mélange) of the two. We discuss this table and the implications of it in Section 5.

<table>
<thead>
<tr>
<th>Form of improvisation</th>
<th>Occurrence and changes over time</th>
<th>Level of improvisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating solutions spontaneously (#10)</td>
<td>From the start of the project and continued throughout; several occurrences</td>
<td>Fluid improvisation</td>
</tr>
<tr>
<td>Revising voluntarily (#2)</td>
<td>At the start of the project but became more structured and ceased; many occurrences</td>
<td>Fluid improvisation</td>
</tr>
<tr>
<td>Discovering and appending (#4)</td>
<td>In the early period but became more structured and ceased; fewer occurrences as project became more structured</td>
<td>Fluid improvisation/Improvisational mélange</td>
</tr>
<tr>
<td>Planning while executing (#1)</td>
<td>At the start of the project and continued for some time; many occurrences</td>
<td>Improvisational mélange</td>
</tr>
<tr>
<td>Altering design features in response to shifting circumstances (#5)</td>
<td>Middle period of the project but continued before becoming more anchored and ceasing; many occurrences</td>
<td>Improvisational mélange</td>
</tr>
<tr>
<td>Changing design practices and adapting to new situations (#6)</td>
<td>Middle period of the project and continued for some time; a few occurrences (compared to “Altering design in response to shifting circumstances” increased anchored improvisation)</td>
<td>Improvisational mélange</td>
</tr>
<tr>
<td>Experimenting with work organization (#9)</td>
<td>Late period of the project and continued for some time; few occurrences</td>
<td>Improvisational mélange</td>
</tr>
<tr>
<td>Channeling knowledge to respond to emerging challenges (#7)</td>
<td>Middle stages of the project but continued before becoming more anchored and ceasing; fewer occurrences as it became more anchored in practice</td>
<td>Anchored improvisation</td>
</tr>
<tr>
<td>Appropriating known practices differently (#3)</td>
<td>Throughout the project, anchored in ongoing practices; multiple occurrences but fewer as the project progressed and new practices became established</td>
<td>Anchored improvisation</td>
</tr>
<tr>
<td>Devising responses to urgent needs (#8)</td>
<td>Late period of the project but became more anchored and ceased; fewer occurrences as it became more anchored in practice</td>
<td>Anchored improvisation</td>
</tr>
</tbody>
</table>

5. Discussion and Implications

In this section, by drawing on the patterns and forms of improvisation identified from the analysis described above, we address our research questions concerning the kinds of improvisation there are in a new systems development project within the context of rigid quality standards, and how these kinds of improvisation unfold over time. We then outline the research implications of these findings.
5.1. The unfolding of forms and levels of improvisation over project time

As discussed in Section 4, ten major forms of improvisation were identified at ImproCo, together with their triggers and their socio-technical context (Table 3). The triggers also reflected tensions, of which we found three sets: following formal plans of action while coping with emerging requirements; allowing flexibility to change while stabilizing the system; and finding imaginative workarounds while enforcing quality standards.

The improvisational responses were anchored in either autonomous reflexivity, ongoing practices or a combination of the two (Figure 3). Only three forms of improvisation were anchored in autonomous reflexivity, with either ongoing practice or a hybrid of the two shaping the remaining forms of improvisation. We illustrated in Figure 3 how the different forms of improvisation followed a pattern. Table 4 summarized the unfolding forms of improvisation and grouped them into three ‘levels’: fluid improvisation, anchored improvisation, and improvisational mélange. We elaborate these improvisational levels below.

5.1.1 Fluid Improvisation

As illustrated in our analysis, some forms of improvisation are more spontaneous (individuals’ autonomous reflexivity) and often seen by others as an innovative and ‘bold’ move, as illustrated in Period 1 of the case study. We call this level of improvisation “fluid improvisation” (see top of Table 4). This includes solo performances and the initial stages of some other forms of improvisation, for example, revising voluntarily and discovering and appending. The triggers for these forms of improvisation are spontaneous in their nature, being based on personal inspiration and serendipity. This was illustrated in Period 1 of the project by Scott working on so-called ‘days off’ in order to find a solution. Improvisation in such cases demonstrated imagination, informed by the developers’ ability to anticipate future requirements, as well as creativity, given the developers’ ability to come up with a solution on most occasions.
5.1.2 Anchored Improvisation

At the bottom of Table 4 are forms of improvisation that are more anchored in established, ongoing practice. These established practices are, in part, embedded in quality management standards such as ISO 9001, and are in the schedules and routinized practices of the systems development project team too. We call this level of improvisation “anchored improvisation”, reflecting the associated structural influences. This also includes appropriating previous practices differently, and other forms of improvisation in the final stages (e.g., responding to urgent needs, responding to emergent challenges). The triggers for the forms of improvisation associated with this level were more systemic; they became stronger as the project became more developed and structured through repeated improvisational occurrences and increasing systemic influences such as stabilizing technology, as per Period 3 of the case study.

5.1.3 Improvisational Mélange

Between the fluid and anchored levels in Table 4 is a cluster of improvisational forms that often start ‘fluidly’, but gradually become anchored in established practices. We call this middle level “improvisational mélange”. The triggers for these forms of improvisation were related to self-motivation and inspiration at the project’s start but subsequently became more grounded in socio-technical context. The developers frequently sought to improvise solutions outside their ‘normal’ work routine (as illustrated by Scott’s work), which helped to expand the possibilities of the hardware and software under development. The amount of improvisational activity seemed to relate to various milestones throughout the project life cycle. Hence, developers’ and managers’ experience of project time, represented by the pressure to meet certain milestones, helped to shape the improvisational forms, mainly at the improvisational mélange level. Anchored improvisation increased as developers were able to evoke prior experiences and to appropriate established practices. The fluid improvisational activities were less in evidence, however, as they became more anchored in structuring practices.
As shown in Table 4, most of the forms of improvisation were at either the improvisational mélange or anchored improvisation levels. While this indicates that many of the forms of improvisation occurred later in the project, the earlier forms of improvisation were qualitatively different – they were based on autonomous reflexivity, for instance, in contrast to the later ones that were anchored on structures such as ongoing practices and procedures (Pavlou & El Sawy, 2010). In the next section, we capture this phenomenon through the 'paradox of progressive saturation'.

5.2 The changing nature of improvisation: A Paradox of Progressive Saturation

Our analysis indicates that as a systems development project progresses, fluid improvisational work becomes increasingly saturated with structural influences such as ongoing practices and procedures. Yet, while the expectation is for work per se to be correspondingly more structured, the increasing project knowledgeability of the developers yields forms of improvisation that are progressively more innovative. An example from the case study is the unconventional means of handling last-minute customer requests despite looming ISO-based sign-offs.

At the beginning of the project, there was greater latitude for autonomous reflexivity because improvisation was only loosely anchored in structural influences; it was more fluid and subjective. In the middle and later periods of the project, as the influence of structural constraints (e.g., structuring processes such as the sedimentation of routine practices and the stabilization of technology features) on the nature of improvisation became stronger, improvisation became more anchored in practices. And yet, the increasing project knowledgeability of the developers yielded forms of improvisation that were progressively more innovative. The developers were, therefore, able to draw on their highly local and timely knowledge to improvise innovative solutions.
As a final illustration of the evolutionary process of improvisation per se and its constituent forms, we conceive this as a paradox funnel – see Figure 4 below. Moving from left to right, we depict that the initial latitude for autonomous reflexivity and fluid improvisation is increasingly saturated by the influence of structural constraints on improvisation. Yet, paradoxically, the increasing project knowledgeability of the developers yields forms of anchored improvisation that are progressively more innovative, which we call a 'paradox of progressive saturation'. It is paradoxical because the nature of improvisation over the course of a systems project reveals seemingly contradictory elements of the improvisation, that is, simultaneously becoming both more saturated (through the increasing influence of structural constraints) and more dynamic (through developers being progressively more innovative by drawing on their increasing knowledgeability). As new systems development projects progress, the nature of improvisation can transform from fluid forms of improvisation to forms of improvisation that are anchored more on ongoing practices and structures. This depicts the changing nature of improvisation over the course of a single systems project within the context of rigid quality management standards.
This conceptualization of the changing nature of improvisation has several research implications. First, our characterization of the nature of improvisation as a paradox of progressive saturation (Figure 4) complements existing views of improvisation paradoxes in the IS literature, such as ‘planned serendipity’ and ‘rehearsed spontaneity’ (cf. Zheng et al., 2007), by adding another dimension to illustrate the tensions and dynamics as different forms of improvisation unfold. Improvisation in systems development literature is often treated as homogeneous within the lifetime of a systems project. Research on improvisation within systems projects can benefit from insights into the dynamics of tensions, triggers and their socio-technical context, and the associated forms of improvisational practice, as well as how these improvisational forms change over the course of a systems development project life cycle, such as the one we have empirically investigated.

Second, insights from the study also contribute to better understanding of the paradoxical tensions between control and improvisational flexibility in systems projects within the context of safety-critical, rigid quality management standards. The literature on ISO 9001 suggests
that the autonomous, ‘creative’ thinking that we observed is discouraged by the imperatives for formal procedures (Benner & Tushman, 2002) and bureaucracy (Singels, Ruël, & van de Water, 2001). However, in our case, an unforeseen outcome of ISO implementation was that it actually forged improvisation. This was enacted through the ingenuity of the individual developers and their very localized culture of experimentation. Their knowledgeability, in terms of project, domain, and technical knowledge, enabled them to enter into particular ‘levels’ of improvisation, that is, more autonomously reflexive or more anchored in practice – there was more latitude for autonomous reflexivity and fluid improvisation earlier on in the project. While the scope for fluid improvisation decreased towards the end of the project, the forms of improvisation became more innovative as the project ran out of time and structural influences took hold. What we found in this paradoxical setting was that the latitude to improvise later on in the project was related more to the accumulated local project knowledge and practices (including the practice of improvising). This was balanced by higher degrees of fluid improvisation earlier on in the process (creating solutions spontaneously, discovering and appending, and revising voluntarily). Therefore, in the context of the study, accumulated knowledge was at a premium later on in the project if the developers were to pull off any form of improvisation at all, which they did.

Finally, our findings also contribute to understanding of the dynamic interplay between improvisation and learning. According to Weick (1998, p. 546), “improvisation does not materialize out of thin air” – prior experiences are key. As the bespoke ImproCo project progressed, the knowledge and learning of the developers grew, contributing to forms of improvisation that were increasingly innovative. Another aspect here is that the actors were able to reflect on progressive ‘lessons’ through their capacity to disembed from routines and break away from structural constraints. This, in turn, helped them to restructure the emergent structural constraints. In every moment of improvisation, there is an opportunity to push the boundaries of what is possible with the existing artifacts and expertise, and to create new accumulations of knowledge (knowledgeability). Different forms of such knowledge are
associated with different levels of improvisation. For example, earlier in the project, the fluid level of improvisation is less dependent on localized knowledge and the structural settings of the project, and more on broader knowledge and personal inspiration. As Moorman and Miner (1998b, p. 703) claim, this level of improvisation discards links between existing practices and “composes new patterns”. That being the case, anchored improvisation later in the project, which is an exploitation of previous practices, depends significantly more on accumulated local knowledge and the recursive occurrence of practices, which helps reproduce and legitimize the knowledge. The accumulated organizational knowledge is, therefore, an assemblage of previous comprehensions that have been laid down during the current and previous projects of the organization. Knowledge creation could be seen as an unintended outcome, because this is not the aim of the improvisational act. This offers a nuanced finding when compared to Miner et al. (2001, p. 331), who argue that improvisation is different from other organizational learning, such as a planned set of procedures to explore and retain knowledge.

6. Conclusions

In this paper, we have conceptualized the changing nature of improvisation over the course of a systems development project. We have identified ten major forms of improvised practice, along with their triggers and socio-technical contexts, in a systems development project within the context of rigid quality management standards. We have argued that these forms of improvisation are anchored in either autonomous reflexivity, ongoing practices or a combination of the two, resulting in three ‘levels’ of improvisation – fluid improvisation, anchored improvisation and improvisational mélange. We have argued that, as the project progressed, the latitude to improvise became constrained by the increasing saturation of structural influences on improvisation. Yet, paradoxically, the increasing knowledgeability of the developers yielded forms of improvisation that were progressively more innovative, which we call a ‘paradox of progressive saturation’.
Our research comes with a number of practical implications. The findings that developers were able to achieve different forms of improvisation throughout the project (with varying success), despite reinforcing the rigid context of planning and quality management, indicates that management cannot solely rely on detailed planning to resolve complex development set-ups. Process management frameworks (such as ISO 9001) generate an overemphasis on tools, techniques, and methodologies, and therefore the prescribed procedures alone can rarely resolve all difficulties (cf. du Plooy, 2002). Improvisational activities help to cope with such difficulties by maintaining flexibility. However, placing sole trust in the improvisational capabilities of developers and managers without planning might cause other problems, such as inconsistency, poor coordination, and quality problems. To retain an innovative capability, firms must maintain the right balance between the support structure (e.g., based on ISO framework) and the latitude for improvisation needed to overcome these problems.

Further, the study has shown that the process management framework of ISO 9001 was sometimes at odds with the improvisational practices at ImproCo, as ImproCo needed to maintain innovative capacity. Nevertheless, the frameworks were seen as essential for maintaining a perception of quality (cf. Nandhakumar & Avison, 1999) and satisfying annual ISO 9001 assessments. Such use of process management frameworks as ‘scaffolding’ seems to allow, nurture and contain some improvisations in the daily work of system development. At ImproCo, the developers and managers followed the process management framework just enough to sustain the ISO certification process, but left enough room to practice a variety of improvisations.

Our research also comes with some limitations. In particular, it builds on a single case study, focusing on improvisation at a project level. Studies in multiple contexts may facilitate cross-comparisons to identify possible variation in how improvisation unfolds over time in other settings. Further, individual project team members’ experiences seemed to differ, yet this study captures this phenomenon only partially. Future research could seek to develop and analyze multiple, overlapping narratives (cf. Brown, Stacey, & Nandhakumar, 2008) of project
members' experiences, which could help to build on this work to develop a richer perspective of how improvisational forms unfold over the course of a project.

References


Marjanovic, O., & Hallikainen, P. (2013). Disaster recovery — New challenges and


Appendices

Appendix 1: Snapshot of the evolving analysis process

<table>
<thead>
<tr>
<th>Practices identified as instances of improvisation</th>
<th>First order categories (activities indicating improvisational forms &amp; changes, triggers, context &amp; duration (time))</th>
<th>Second-order themes: Levels of improvisation</th>
</tr>
</thead>
</table>
| Extracts from empirical data
Field notes:
The graphical interface was not part of the initial idea or request but they saw it as important
and developed it from scratch as an auxiliary solution.
Jack and Mike developed the interface carefully, but admitted to applying some 'shortcuts' for
testing purposes. However, they said that those 'shortcuts' did not become part of the official releases.
Interview quotation - Jack:
“For me there was no doubt that we would need a graphical interface because the users of this electronic device would like to have a simple solution which does not require additional training. However, the initial idea did not involve any user-friendly approach, so we needed to adapt to that principal requirement.”
[...]
[...]
Planning while executing
- Environmental turbulences, team reflecting and responding to vague specifications vs formal plan [trigger]
- To be interactive, had to envision and anticipate the features [context]
- Many occurrences at the start but overtime ceased (time)
Revising voluntarily
- Self-perceived need to deliver proper solutions for the customer vs standard routines, ISO9100 [trigger]
- Became more knowledgeable of customer needs as the hardware and software were still malleable [context]
- Several occurrences throughout the project
Appropriating known practices differently
[...]
Discovering and appending
[...]
Altering design features in response
[...]
Changing design practices and adapting to new situations
[...]
Channeling knowledge to respond to emerging challenges
[...]
Devising responses to urgent needs
[...]
Experimenting with work organization
[...]
Creating solutions spontaneously
[...]

Diagram showing relationships between improvisation levels.