Scandinavian Journal of Information Systems

Volume 25 | Issue 2 Article 3

12-31-2013

Evaluating End User Development as a Requirements Engineering Technique for Communicating Across Social Worlds During Systems Development

Fredrik Karlsson
Örebro University, fredrik.karlsson@oru.se

Karin Hedström
Örebro University, karin.hedstrom@oru.se

Evaluating End User Development As a Requirements Engineering Technique For Communicating Across Social Worlds During Systems Development

Fredrik Karlsson
Örebro University, Sweden
fredrik.karlsson@oru.se

Karin Hedström
Örebro University, Sweden
karin.hedstrom@oru.se

Abstract. Requirements engineering is a key activity in systems development. This paper examines six systems development projects that have used end user development (EUD) as a requirements engineering technique for communicating across social worlds. For this purpose, we employed the theoretical lens of design boundary object in order to focus on functional and political ecologies during the development process. Four features were investigated: (1) the capability for common representation, (2) the capability to transform design knowledge, (3) the capability to mobilise for design action, and (4) the capability to legitimise design knowledge across social worlds. We concluded that EUD means a high degree of end user involvement and takes advantage of end users' know-how. It has the ability to capture requirements and transfer them into the final information system without the need to make an explicit design rationale available to the systems developers. However, systems developers have little or no influence on business requirements. Their role is mainly as technical experts rather than business developers. The systems developers took control and power of technical requirements, while requirements that relate to business logic remained with the end users. Consequently, the systems developers did not act as catalysts in the systems development process.

Keywords: End user development, requirements engineering and systems development.

Accepting editor: Samuli Pekkola

1 Introduction

Requirements engineering has been recognised as a key activity in creating or altering information systems (Ocker et al. 1996), i.e. in systems development. It has been recognised that errors and misunderstandings during the requirements engineering phase is one of the greatest single causes for project failure (Hansen et al. 2008). Consequently, the way in which this phase is carried out has significant impact on the final information system. At the same time, this process tends to be error-prone inasmuch as systems development involves finding a solution to the paradox of 'wicked' problems (Rittel 1984) – in other words, you have to 'solve' the problem before you can clearly define it, and then design a workable solution. The search for approaches that meet this challenge has been extensive (e.g., Checkland 1981; Martin 1991; Schwaber and Beedle 2001; Kruchten 2004), and has led to a greater understanding of this issue.

Today, requirements engineering is viewed as a continuous 'decision-rich problem solving activity' (Aurum and Wohlin 2003) during systems development. It means that requirements are not found, but designed by system developers together with end users in analogy with architects in the building industry, 'who produce design sketches based on brief specifications from their clients, and who agree then what should be built in terms of these sketches' (McDermid, 1994). These sketches and the requirements are then refined in an iterative pattern in order to decide on how to realize the ideas. Consequently, requirements specifications evolve through a set of design decisions on what to build.

A large range of different requirements engineering techniques is now available. These techniques aim to structure the decision-making process, and denote the driver behind these decisions. Proponents of participatory design (Mumford 1981; Greenbaum and Kyng 1991; Ehn 1993; Kensing and Munk-Madsen 1993) have argued that people who are supposed to use the final information system (i.e. end users) should participate in this process. Some of these techniques have been developed through research whilst others have evolved as 'best practices' through acquired industrial wisdom.

In this paper we address the industrial practice of using end user development (EUD) as a requirements engineering technique, something that we have encountered in several empirical cases. EUD grew in popularity during the 1990s (Brancheau and Brown 1993; Taylor, Moynihan et al. 1998; Moore 2002), when end users were able to use powerful desktop tools, such as spreadsheets and easy-to-use databases, for development of local information systems. This kind of development is closely linked with the daily work carried out by end users. Indeed, Brancheau and Brown (1993) defined EUD as: "the adoption and use of information technology by personnel outside the information systems department to develop software applications in support of organizational tasks." Existing research has shown that the results of EUD are not without controversy (e.g., Karten 1990; Panko 1998; McGill and Klobas 2005). However, relatively little is known about EUD when used in systems development as a requirements engineering technique (Rittenberg and Senn 1993; Dodd and Carr 1994). Thus, we pose the research question: what are the advantages and disadvantages of using such a practice in systems development to communicate requirements across two social worlds?

The aim of this paper is to examine EUD as a requirements engineering technique for communicating across social worlds (groups of actors with shared knowledge, interests and tasks) in

systems development. For this purpose, we employed the theoretical lens of design boundary object (Bergman et al. 2007). Design boundary objects are a specific type of boundary object (Star and Griesemer 1989) used to communicate design knowledge across social worlds in order to propel the systems development process forward. Four aspects of design boundary objects were used to study the uses of EUD in six case studies: (1) the capability for common representation, (2) the capability to transform design knowledge, (3) the capability to mobilise for design action, and (4) the capability to legitimise design knowledge across social worlds. We show, through empirical examples, that EUD can be an effective technique for incorporating end user's know-how in the final design, thus addressing the tacit dimension of knowing (Polanyi 1983). However, this technique changes the role of systems developers, who no longer act as catalysts in the systems development process. This research therefore advances existing research about requirements engineering techniques (Kensing and Munk-Madsen 1993; Maiden and Rugg 1996; Malcolm 2001) and the pros and cons of these techniques.

The remainder of this paper is structured as follows. The next section introduces the theory of design boundary objects, before going on to discuss existing research on EUD. In the fourth section, we present our research setting and research method. The fifth and sixth sections describe our analysis, focusing on one of the case studies. The seventh section comprises a discussion of the research findings, their implications for practice and research, research limitations, and ideas for future research. Finally, the paper draws several conclusions.

2 Theoretical lens: design boundary objects

Design boundary objects (Bergman et al. 2007) are artefacts, such as prototypes, storyboards and use cases, which are generic enough to facilitate design cooperation across social worlds, but still specific enough to allow diversity in interpretation. For example, in the field of requirements engineering, a use case carries information from the end users to the systems developers about the order in which specific tasks must be carried out. However, the exact interpretation of those tasks might still be different depending on the background knowledge of both the end users and the systems developers. Consequently, design boundary objects are similar to the more generic boundary objects (Star and Griesemer 1989; Star 1993), where the latter are any artefact shared by two or more actors from different social worlds. Boundary objects have often been used to study coordination activities across multiple social worlds. These include different types of objects such as museum artefacts (Star and Griesemer 1989), engineering drawings (Henderson 1991) and patients' medical records (Berg and Bowker 1997). Few studies have focused more specifically on design boundary objects, i.e. the role of boundary objects in design processes. Bødker (1998) described design representations as 'containers' of different ideas that have limited capability to communicate between different social worlds. Bergman et al. (2007) showed that design boundary objects have an impact on two ecologies in an organisation: the functional and the political. The role of the functional ecology is to investigate how technologies can support and extend an organisation's work processes. The political ecology addresses power structures within an organisation, determining who can make design decisions and when these decisions can be made. Design boundary objects have four features that determine how each

object functions within these ecologies. Table 1 shows the features and how we define them in this paper.

Requirements engineering techniques guide the navigation of design paths across functional and political ecologies. For example, user stories or prototypes are requirements engineering products that connect design routines across multiple social worlds; as such, these artefacts become design boundary objects. These artefacts place a different emphasis on the above-mentioned features of design boundary objects. They have different advantages and disadvantages in helping actors to overcome gaps in the functional and political ecologies. In this paper, we have a specific interest in end user developed information systems as design boundary objects in systems development. We investigate their advantages and disadvantages in helping to overcome design knowledge gaps residing in the functional ecology and helping to resolve agreement gaps in the political ecology. We will next discuss existing research into EUD, showing that EUD has, to a limited extent, been discussed in relation to design boundary objects in systems development.

Features	*Definition*
Promote shared representation	The design boundary object encapsulates a shared functional and/or a political form through common syntax and semantics, which is understood across two or more social worlds.
Transform design knowledge	The design boundary object combines and transforms representations that move towards finding a feasible functional and political solution.
Mobilize for design action	The design boundary object invites action into the functional ecology in order to reduce problem or solution ambiguity in a design, and/or the political ecology to participate in decision making and resource allocation.
Legitimize design knowledge	The design boundary object legitimises content: functionally, the content needs to be legitimised as correct and valid design knowledge in order to support and extend the organisation's work processes. Politically, the content needs to demonstrate the acceptance of goals, problems and solutions by power structures within the organisation in order to justify the movement of design knowledge across social worlds.

Table 1. The four features of design boundary objects

3 EUD research

EUD does not involve any distinct activities (Ko et al. 2011). Rather, it forms part of the end user's daily work and the developed information systems evolve in small increments over time in an iterative pattern. End users often use fourth generation development tools (Sayles 1990), tailorable information systems (Eriksson and Dittrich 2007), or Web 2.0 tools (Costabile et al.

2008; Cappiello et al. 2013) to frame the problem and search for solutions. For example, Floyd et al. (2007) reported on how end users construct mash-ups using Web 2.0 and public available APIs (Application Programming Interfaces) in order to solve problems in their every day lives. The development pattern that they described is iterative and hardly ever controlled, as promoted in systems development methods. Instead, they described how the mash-ups evolved based on the end users' current understanding of the problem, with problem solutions evolving to meet changing needs.

EUD is often viewed in terms of development activities that are independent of systems development. In existing EUD research, end user developed information systems are usually not studied as design boundary objects. Instead the focus is on the EUD process and the consequences of using these artefacts in organisations' work processes (e.g., Davis 1988; Teo and Tan 1999; McGill 2004). Hence, these studies do not focus on end user developed applications as part of systems development projects. However, a few notable exceptions do exist. One such example is Dodd and Carr (1994), who proposed the concept of Systems Development Led by end Users as a cross between systems development and EUD, where end users act as project mangers. Thus, it is not a full integration between EUD and systems development, since the end users did not develop artefacts on their own. However, Dodd and Carr (1994) theoretically illustrated the possibility to use end user developed information systems as starting points for projects when they discussed development efforts as a continuum between systems development and EUD. Still, they provided no such empirical evidence. Rittenberg and Senn (1993) discussed the repackaging of end user developed applications as full-blown systems that act as a disseminating mechanism in organisations. They addressed several interesting aspect such as ownership of applications and reliability issues. Still they did not go into details on how repackaging of end user developed applications affect the development process. For example, they did not discuss the effects on the political ecology. Floyd et al. (2007) discussed the potential of end user developed mash-ups as a way to show to professional developers what to develop, i.e. as requirement engineering tool. However, they have not analysed the developer-side of the process. We can conclude that none of these studies focused on how EUD affects both the functional and political ecologies in systems development.

Our study of existing research also revealed at least two aspects that are important when discussing end user developed information systems as design boundary objects. Both aspects seem to concern the capability to transform design knowledge and mobilise for design actions as well as the capability to legitimise design knowledge. The first aspect is the quality of the requirements, which in our research is viewed as an integral part of the end user developed information system. Since EUD is an intertwined process, requirements evolve over time (Costabile et al. 2006). Consequently, using the end user developed information system as a requirements engineering artefact means that it is volatile (Ko et al. 2011), especially in its early stages when the end user explores the problem domain.

The second aspect is the quality of the proposed design. Davis (1988) argued that the use of external analysts, i.e. systems developers, leads to better designs. The systems developer normally contributes with expertise knowledge, acting as an expert or facilitator during systems development (Mogensen 1992). Often, end users become aware of the design implications at a later stage, through a trial-and-error process (Ko et al. 2011). The quality aspect has been explored in a number of studies (e.g., Ditlea 1987; Teo and Tan 1999; Panko and Halverson

2001; McGill and Klobas 2005; Karlsson 2008) and substantial error rates have been found, both when students and professionals have developed end user applications. These findings are important since they indicate that quality problems can be expected when EUD is used as a requirements engineering technique, that may lead to a) large number of redesigns during the development process, and/or (b) poor quality in the final product. However, Kreie et al. (2000) concluded that the quality of implemented models can be improved with training in systems analysis and design methods. For example, Burnett et al. (2004) proposed an interactive testing method to support the development of spreadsheets by end users and McDaid (2008) argued for test-driven EUD. Other scholars proposed dealing with lack of skills through, for example, the enforcement of best practices for spreadsheet design (Powell and Baker 2003).

In summary, few existing research studies have examined the use of EUD in combination with systems development. Hence, it is difficult to assess EUD as a requirements engineering technique in systems development in terms of existing literature. However, it is possible to conclude that the quality of end user developed applications has been debated extensively over the years. To some extent, this could have a bearing on potential quality problems when using them for requirements specifications.

4 Research approach

We carried out six case studies in order to explore end user developed information systems as design boundary objects in requirements engineering. The study design can be classified as an interpretative case study (Yin 1994; Walsham 1995; Klein and Myers 1999; Goldkuhl 2012), that allowed us to analyse the functional and political ecologies (Bergman et al. 2007) of our projects.

4.1 Research setting and cases

We studied six systems development projects in four different companies following guidelines for theoretical sampling (Yin 1994). Hence, these projects were selected both for their similarities and differences. All four companies used EUD from time to time as a technique for requirements engineering, although they employed well-known systems development methods for the remaining project activities. The precise reasons for these choices varied slightly between the actors. However, they can be summarised as a general need to elicit good ideas from highly specialised business processes where business knowledge is regarded as being difficult to communicate. Several earlier projects had failed because of this problem, and end users had therefore started to use "home-built" prototypes to illustrate their ideas. Another reason was the fact that several of these projects were carried out by people working in different physical locations, which complicated communication even further. The EUD was carried out using spreadsheets, which is the most commonly used tool for this type of development (Burnett et al. 2004).

Case no	*Type of information system*	*Industrial partner*	*Business complexity*	*Man-hours*	*No of developers*	*No of end users*	*Systems development method*
1	Product calculation	1	High	500	3	1	XP
2	Investment analysis industrial equipment	1	Medium	300	2	1	XP
3	Price analysis and simulation	1	High	1600	3	1	XP
4	Product calculation	2	Medium	500	2	2	DSDM
5	Investment analysis industrial equipment	3	High	700	2	2	XP
6	Product calculation	4	Low	200	2	3	Crystal Clear

Table 2. Overview of cases

Table 2 presents a compilation of the six cases. The leftmost column shows the case number. The second column from the left shows the type of information systems that were implemented. The remaining columns contain, in due order, the identifier of the industrial partner, the level of business complexity, the number of man-hours spent on each project, the number of developers, the number of end users, and the systems development methods in use. From the table we can see that the six cases differed in terms of business complexity. In cases with high business complexity, the information system is based on knowledge about the business' core process. Such knowledge can: a) be difficult to articulate or it is something that the end user(s) has learned over many years, and b) be something with which the systems developer(s) is unfamiliar. In other words, these projects posed a greater challenge with regards to communication between social worlds than projects with medium or low business complexity. The projects also differed in size, both with regards to man-hours spent by the systems developers on the project and the project team size (number of developers and end users).

Due to space limitations it is impossible to provide a detailed description of the analysis of all six cases. Therefore, our first case is used as the main case in the analysis and the others are used to illustrate conflicting findings. The first case was chosen based on its explanatory power

to illustrate the advantages as well as the disadvantages of EUD as a requirements engineering technique for communicating across social worlds in systems development. Below we give an introduction to all six cases. For the first case we provide additional details in order to give the analytical context.

Background to case one

The first three cases were taken from an international manufacturing company engaged in metal cutting. The first case was the most innovative of the three projects that involved this particular industrial partner. The aim was to develop a product calculation system for the metal cutting industry that could provide quick price quotes when talking to customers. This task poses two major challenges, both of which are related to the complexity of the calculation. First, the cutting process itself contains a large number of cutting machine parameters, which have to be taken into consideration. Second, the complexity of the piece of metal that needs to be cut affects the calculation; it requires the customer to provide a technical drawing showing the exact measurements needed for the calculation. The project's goal was to create an information system where the customer only needed to answer a limited number of questions, all of which could be asked over the telephone. The proposed solution had the potential to reduce the information flow between the customer and the supplier as well as reduce the time needed to provide the customer with a quote. A vision statement defined the scope of the project as: 'A product calculation system that does not require access to customer drawings in order to answer questions about price when talking to the customer.'

The development team consisted of four members: three systems developers and one end user. The end user had 22 years of experience in the field of industrial cutting. His daily tasks involved sale of cutting machines, equipment and consumables. The end user's experience with the EUD tool Microsoft Excel was excellent. It is part of his daily work and he has, for example, developed a budget system using this tool. In addition, the end user had a good general knowledge of computers.

The end user belonged to one of the company's regional sales offices in Europe, which in this specific region consisted of seven people. It meant that he daily interacted with customers to identify their needs and quote on new solutions. This gave him very good knowledge about the actual challenges that the company's customers were facing. Internally, the end user worked together with the other employees at the regional sales office and the product engineers at the factory, in order to quote unique solutions to specific customers. The end user reported to the regional sales manager, who guaranteed the end user's time on the project. The information system developed in the studied project became part of offering unique administrative sales support related to the quoted machines.

The systems developers belonged to the company's administrative organisation, and constituted a separate department of approximately 50 people. Subsequently, this was the social world of the three systems developers taking part in the second project. The systems development department was not located in the same country as the end user's regional sales office. Project teams were formed based on the current project portfolio and the competences needed. It was common that systems developers worked in several parallel projects at the same time, which

meant they often reported to more than one project manager. Each project manager reported to the head of the systems development department, who allocated systems developers to the projects. The systems development department's responsibility was to build and support both administrative and embedded (e.g CNC control software) information systems. It meant that the systems developers were specialized to work with one of the two categories. The current project was viewed as an administrative information system by the head of the systems development department, although it contained technical parts about cutting processes. However, it was not viewed as something being part of a cutting machine. Hence, the systems developers were not highly experienced on cutting processes. At the time of our study the company employed an internal 'buy-sell market', which meant that different departments could buy systems development services from the systems development department. This was the case with the current project, which was funded by the regional sales office based on the decisions of the regional sales manager.

The systems development method employed by the systems development department was eXtreme Programming. The project was allocated a timeframe of 500 man-hours during the course of six calendar months. The project consisted of four iterations: one EUD iteration and three systems developer iterations in which the end user requirements were implemented. The latter iterations were closed out by end user reviews and lessons learned were fed back into the process for redesign during the next iteration. The EUD iteration was not as strict as the systems developer iterations. The EUD prototype was continuously developed, designed, tested and redesigned using small incremental steps over the period of a month and a half. The end user did not view the redesign in terms of iterations.

Background to cases two to six

The second case concerned an investment analysis system of heavy industrial equipment. The goal of the developed information system was to ease the burden of the sales force when working with customers to make initial calculations for the purchase of a new cutting machine. The project was initiated by the end user, based on his experiences in quoting for machines. The project was allocated two systems developers and a timeframe of 300 man-hours over a period of seven calendar months. The project resulted in the development of an information system that is currently used in several countries.

The third case was intended for the industrial partner's internal use. One major challenge at that point in time was price analysis and simulations. Since the company was to operate in several markets, this task was particularly time consuming. The developed information system simplified this task, and included the integration of three enterprise resource planning systems. The end user initiated this project because of the problems he had experienced. The three systems developers had 1,600 man-hours at their disposal over a period of four calendar months. Although the final information system has a limited number of users, it has reduced the time for completing this price analysis and simulation task by 90 percent.

The fourth case was a project from the plastics industry. The company wanted to modernise their method of product calculation. At that point in time they had a manual and no unified way of making product calculations. Hence, two experienced product managers were given the

Published by AIS Electronic Library (AISeL), 2013

task of designing a spreadsheet-based version of their way of working. Two systems developers were assigned to convert the end users' prototype into a final information system. Their timeframe was 500 man-hours over a period of three calendar months. The calculation system is currently being used in parallel with manual calculations.

The fifth case was a project from an international manufacturer of process industry components. It shares characteristics with the second case, since both concern investment calculations. However, in this case the system was not used in the company's dialogue with the customer. In addition, the calculations were more complex and involved a very large number of decisions relating to different variants of the quoted solutions. Hence, producing a quote for this type of equipment required detailed business knowledge. The aim of this project was to create a unified way for the company to quote for new equipment, since they had experienced profitability problems with particularly complex customer projects. In addition, they wanted to increase the number of people that could offer quotes for this type of equipment. Two experienced area sales managers initiated this project and drove the development of a framework for investment calculations. This framework was implemented in Excel and then converted into a web application by two systems developers. To complete the task, 700 man-hours over a period of four calendar months were required. The system is currently in use, although it has not increased the number of people who are able to quote for new equipment.

The sixth case was the project with the lowest level of business complexity. This case involved a national subcontractor of metal parts, which was experiencing problems with the many different ways that product calculations were made within the company, particularly in terms of profitability assessments. Three end users were therefore assigned to the task of developing a system for product calculation that would be generic enough to cover the complete product range and could be used throughout the organisation. Two consultants used 200 man-hours over three calendar months to implement the final system.

4.2 Data sources

Data sources included log books, semi-structured interviews, information systems created by the end users and the systems developers, and observations. This triangulation of several data sources provided multiple perspectives (Patton 1990) on the development processes, and allowed for validation of design decisions and knowledge transfer. We were in particular interested in tracking a) the design decisions that the end user expressed through the end user developed information systems, since these systems were part of the projects' requirements specification on what to build, and b) what effects these decisions had on the functional and political ecologies. One such example, from the first case, is the end user's requirement that the cost of capital has to be included when calculating prices for cutting a metal plate. The end user expressed this requirement when including an algorithm for calculation of capital cost in the end user developed information system.

Log books. The project team members, i.e. end users and systems developers, were instructed to maintain log books. The content of these log books included anything considered by the actors to be major design decisions on what to build, fulfilment of requirements, arguments and the way in which they had been communicated. They provided an effective means

for cross-checking informants' interviews, the design decision taken, and any knowledge that was transferred in order to build the final information system based on the end user developed application. In practice, the log books were used as a starting point when creating the interview guides (Patton 1990). Identified design decisions were added to the guides for discussion.

Semi-structured interviews. 26 semi-structured interviews were carried out with the end users during various stages of the projects. We used the end users' log books and the functionality of the end user developed information systems as input for the interviews. Questions addressed the design decisions described in the log books, the design rationale behind them, and the types of communication that occurred between the end user(s) and the systems developer(s). This provided the end users' perspective on the development processes.

In addition, 37 semi-structured interviews were carried out with the systems developers during the projects. The purpose of these interviews was to capture design decisions from the systems developers' perspective. The interviews with the systems developers were developed from the same collection of material as the end user interviews: the end users' and systems developers' log books, the functionality in the end user developed information systems, and the final information systems.

Information systems. We also obtained access to the information systems developed in the projects. This access included both end user developed versions (i.e. the EUD prototypes) and the deployed information systems (including the code bases). They provided an effective means to validate the informants' log books during data collection and to identify functionality that had not been mentioned in these documents. In particular, it allowed us identify: a) if end user designed functionality had been implemented in the deployed information system, and b) if the deployed information system was based on the end users' design solutions and, if so, to what degree.

Observation. We participated as observers in 42 project meetings, where we followed the progress of the projects. Team interactions included face-to-face meetings and teleconferences. This participation gave us first-hand information of the projects and the type of discussions that took place during these meetings – mostly focusing on project progress reports. It allowed us to capture data about any design decisions that were taken during these meetings.

4.3 Case analysis

The four features of design boundary objects and the two design ecologies laid out by Bergman et al. (2007) were used during our analysis. For that purpose we tailored the theoretical lens to fit our research question: what are the advantages and disadvantages of using EUD in systems development for communicating requirements across two social worlds. Our operationalisation of the theoretical lens is presented in table 3.

The table has three columns. The leftmost column contains the four features of the design boundary object framework: 'Promote shared representation', 'Transform design knowledge', 'Mobilise for design action' and 'Legitimise design knowledge'. The second column shows that each feature has two ecologies: the functional and the political. The rightmost column shows our analytical questions for each ecology and the feature that it belongs to. Hence, this column

Published by AIS Electronic Library (AISeL), 2013

contains an analytical framework of total eight focal areas that we have applied on our empirical data.

The analysis was carried out in three steps. In the first step we listed unique design decisions. Second, we sorted these design decisions into two categories – decisions made by the end users (on what to build) and the systems developers (on how to build) – from within our two selected social worlds. Based on these design decisions, we reconstructed the different development processes found in the project and the way in which decisions in the two social worlds built upon each other. As a third step, we analysed the design decisions using the four features of design boundary objects and two design ecologies, i.e. using the analytical questions presented in the rightmost column of table 3. During the analysis we looked for patterns that either fully supported, partially supported, or did not support the four features of design boundary objects. This step was carried out in an iterative pattern in order to cover all identified design decisions.

5 Analysis of EUD prototypes in the functional ecology

5.1 Promote shared representation

In support of a shared representation. The shared representation within the EUD prototype consists of the graphical user interface, algorithms, data used as input in the prototype, and data produced by the prototype. Support for a shared functional representation was found in four of the six cases. Our analysis of the first case shows that from the end users' perspective, the EUD prototype was 'a workable solution, that I have used extensively during customer meetings'. Thus, the EUD prototype contained the functionality needed for calculating the cost of cutting any shape from different types of materials. From the systems developers' perspective the artefact consisted of enough technical details to clarify the functionality as chunks of requirements. These chunks were identified through reengineering the EUD prototype. One of the systems developers stated: 'We spent much time defining different paths through the system, which we then treated as user stories.' Based on their view of the EUD prototype's functionality, they defined work tasks for the forthcoming development work.

One example of how the two social worlds interpreted the EUD prototype is the functionality required to calculate the cost of piercing through the plate, which is one component of the total cutting cost. The end user described this calculation as a cost affected by 'the thickness of the plate, the consumables and power source used and of course if you are cutting aluminium, carbon steel or stainless steel'. In other words, a detailed account that uses business-specific concepts is needed to ascertain how the different parameters affect the piercing cost. The systems developers on the other hand described the same functionality as 'an algorithm using these input values', referring to specific spreadsheet cells containing data about plate thickness and types of consumables. Their perception of the same functionality was described using more generic systems development and programming concepts.

Features	*Design Ecology*	*Description and analytical question*
Promote shared representation	Functional	How the requirements engineering artefact contributed to sharing design knowledge between the different social worlds, i.e. to what extent do the two worlds have a shared view of the requirements and the design (what to accomplish)?
	Political	How the requirements engineering artefact contributed to perspective sharing between different social worlds, i.e. to what extent do the social worlds have a shared view of why certain requirements and designs are conveyed?
Transform design knowledge	Functional	How the requirements engineering artefact contributed to finding feasible functional solutions in the recipient social world, i.e. to what extent did the artefact provide sufficient information for finding a refined solution?
	Political	How the requirements engineering artefact contributed to the handing of control and power to the recipient social world, i.e. what functional solutions need to be refined?
Mobilize for design action	Functional	To what extent the requirements engineering artefact contributed to solving the needs of the recipient social world, i.e. to what extent did the artefact provide sufficient information for implementing the refined solution?
	Political	How the requirements engineering artefact contributed to the recipient social world accepting the design, i.e to what extent did the recipient social world mobilise resources based on the expertise residing in the provider world?
Legitimize design knowledge	Functional	How the requirements engineering artefact contributed to verifying and validating the usefulness and correctness of the solution, i.e. to what extent has the artefact been used to verify and validate the information?
	Political	How the requirements engineering artefact contributed to demonstrate acceptability of design decisions so as to authorize the movement of design knowledge across social worlds, i.e. to what extent has the artefact been used in the decision-making processes?

Table 3. Operationalisation of the theoretical lens

The EUD prototype had enough plasticity and sufficient detail for describing the requirements. In particular, it contained sufficient detail to share a technical understanding of the requirements within the functional ecology. At the same time, the artefact had enough plasticity to allow both social worlds to discuss the same functionality, using concepts with which they were familiar.

In partial support of shared representation. During the third and fifth cases the systems developers had initial problems grasping the complete requirements expressed by the EUD prototypes. It is evident from the interviews with the systems developers that they initially had difficulties sorting out the functionality and the way in which it was constructed. With regard to the fifth case, the end users had used Visual Basic for Applications (VBA) in Excel to automate certain functionality in the EUD prototype. For example, they used VBA-code to restrict which process components that were possible to combine when quoting equipment. The functionality was difficult to dissect because of the many interdependencies that existed in the VBA-code. One systems developer working on this project said: 'you can tell from the code that it has evolved as the needs have occurred, without any consideration of the big picture. And it is not so simple to disentangle'. Similar thoughts were shared by one of the systems developers from the third project: 'the sheer size of code makes it a challenge to understand, since both the quality of the code and the documentation were so-so. I guess parts of the code has been automatically generated [by Excel], and then they have copy-pasted it with small modifications'. Consequently, the systems developers had to spend more time than expected on learning the EUD prototype to get a picture of what to construct.

5.2 Transform design knowledge

In support of transforming design knowledge. From the functional ecology perspective, we found that the EUD prototype supported the transformation of design knowledge in all six cases. In the first case, the EUD prototype bridged the gap between the project team's initial vision and the end user's requirements. To a large extent, the way in which these requirements were met in the final information system was non-verbalised. The EUD prototype consisted of a set of functionalities that the systems developers refined for usability and technical implementation. For example, based on the tables of data on gas consumption, preheating time, machine settings and so on that are found in the EUD prototype, the systems developers created a conceptual data model of the new information system. This model has a major technical design advantage over the EUD prototype as it offers a greater reduction in redundant data. One of the systems developers summarised this as: 'it was easy to see how we should proceed. When we first saw the tables we naturally thought of a database, but actually we ended up using separate files that are loaded into the application. But all the tables concerning different power sources shared the same structure, and therefore became one table. One can say that the end user's solution contained data redundancy'.

5.3 Mobilise for design action

In support of mobilisation for design action. The EUD prototype contains a graphical user interface, algorithms, data used as input in the prototype, and data produced by the prototype, all of which were mobilised for design action by the systems developers. The work tasks defined by the systems developers gradually transformed the EUD prototype into the final information system. For example, during the first case study a new graphical user interface was designed using the analysis of the EUD prototype as a starting point. Data structures were implemented using the tables of data on gas consumption, preheating time, machine settings and so on as models. Calculations, presented as algorithms in the EUD prototype, were rewritten in the programming language of choice. Data derived from the EUD prototype were used as test data. The systems developers stated that, with regard to the main identified work tasks, the EUD prototype was satisfactory in helping to solve their needs. One of the systems developers described the EUD prototype as: 'more detailed requirements compared to a user story'. Furthermore, the interviews revealed that systems developers felt they had a solid ground for mobilisation: 'I feel that the requirement changes are not as frequent, since the user has worked on and tested the concept'.

In support of no mobilisation for design action. During the fifth case study the systems developer experienced some problems when mobilising for design action. The interviews with the systems developers revealed that 'they [the end users] underestimated the complexity' of the information system under development. The user interface suggested by the end users became cluttered when transferred to the full-scale application. The main reason was the amount of data that was made available in the final information system. This problem was not taken into consideration since this requirement was not apparent from the EUD prototype. Furthermore, it was not communicated between the end users and the systems developers when discussing the EUD prototype. One of the systems developers commented on this problem: 'I realize why they [the end users] did not encounter the problem. They only tested with a fraction [of the data] … I would have preferred that we had been more involved concerning this aspect'.

5.4 Legitimise design knowledge

In support of legitimising the design action. The functional usefulness and correctness of the EUD prototypes were legitimised through real world testing during the end users' construction of the artefact. In the first case study, the interviews with the end user revealed that he used the EUD prototype during his daily contact with customers. Thus, he received continuous feedback on his application and was able to make gradual improvements to it. The crucial design decisions focused on what questions the customer needs to be asked without access to customer drawings of the shape to cut. During the interviews, the end user clarified that he 'started out with a larger set of questions to ask [the customer], but I learned which questions I really need. And then I changed the programme'. In addition, he also learned 'what information the customer has and does not have'.

The correctness of the EUD prototype was evaluated by the end user using: a) post-term calculations of real cutting costs, and b) comparisons of calculations that were based on cus-

Published by AIS Electronic Library (AISeL), 2013

tomer drawings of the shape to be cut. According to the end user, the EUD prototype provided 'sufficiently good precision according to my customers, especially considering that they do not need to bother sending any drawings'.

6 Analysis of EUD prototypes in the political ecology

6.1 Promote shared representation

In partial support of shared representation. A shared political view of the requirements was not obtained in the studied cases. The sixth case study, however, stands out and is a near likeness of a politically shared representation. In this case, the systems developers were able to explain 27 of the 45 design decisions made not only from a technical point of view, but also from a business perspective. When comparing this project with the others it was the smallest project of the six, with a business logic that was familiar to the systems developers. This system was a generic calculation system designed to fit several types of products. Hence, similar calculations can be found in product calculation textbooks. Moreover, it contained few complicated formulas and a quite small amount of data compared with the first case that involved a highly specialised and innovative type of product calculation, with which the systems developers were unfamiliar.

In support of non-shared representation. A non-shared representation was the prevailing impression after analysing the six cases. As discussed above, the systems developers had good functional understanding of what to accomplish during the first case study. However, the same developers were, to a large extent (61 of the 68 design decisions), unable to explain the rationale behind the end user's design decisions from a business point of view.

For example, one important design decision proposed by the end user was to include gas consumption tables for a selection of cutting nozzles. This type of table determines the gas consumption needed to pierce a hole in the plate using a specific cutting nozzle. The end user explained this as important functionality to include, since 'you use one type of gas, such as propane, to heat the metal and then use oxygen to burn away the material from the cut. The speed and gas consumption you achieve is determined by the shape and the quality of the cutting mouthpiece'. Hence, the choice of cutting nozzle affects the gas consumption and indirectly the cutting price. Politically, this design decision was based on a number of standpoints. First, the end user made a selection of cutting nozzles that were possible to combine with certain types of material. This selection was based on the end user's knowledge about the quality achieved through combining a certain type of cutting nozzle with a certain type of material, and how this quality stands up against industry standards. Second, only cutting nozzles from certain vendors were selected, based on decisions made by the sales organisation. This selection was made for competitive reasons, in order to promote certain brands. From the interviews with the systems developers it was evident that they did not know these political reasons. The following statement shows the system developers' unawareness of the political side of the design decision: 'the choice of a nozzle depends on your cutting material. Therefore you need to have a broad range of noz-

zles for the user to choose from, to get functioning gas consumption. But the tables are only large chunks of data to me', and 'these are gas consumption tables for different cutting nozzles. I see that it [the algorithm] uses different data from the tables, but I cannot say why'.

6.2 Transform design knowledge

In partial support of transforming design knowledge. From a political ecology point of view, the EUD prototypes only partially allowed the systems developers to take control of the refinement of requirements. The use of an EUD prototype meant that the end user handed over control of requirements related to usability and technical implementation. However, very few design decisions were made in which the systems developers refined the business logic in the final information system.

An example taken from the first case study illustrates this situation. It concerns the tables containing gas consumption data. Control and power were handed over, but only with regard to the technical implementation. One systems developer stated that: 'the table content is, according to the end user, based on the company's own measurements'. Thus, the systems developers accepted them. However, as discussed above, the systems developers questioned the redundant solution found in the EUD prototype, and they provided an improved the solution for storing this type of data. In addition, they questioned that this data was unprotected when it was 'so fundamental for making it [the EUD prototype] work'. Thus, the systems developers suggested that the files to be protected and, later, they encrypted the data files.

6.3 Mobilise for design action

In support of mobilisation for design action. Politically, the EUD prototype did not cause any problems when used to allocate resources during four of the analysed projects. In the first case study, the systems developers used the EUD prototype to mobilise resources to their three iterations, i.e. their release and iteration planning. As discussed earlier, the systems developers treated 'different paths through the system' as being equivalent to user stories. The systems developers used user stories derived from the functional prototype to sign up for different work tasks, and to estimate the time it would take to implement each task. One system developer stated that: 'in many cases the [EUD] prototype has its upsides, since the requested functionality is presented in great detail. Often it makes the assessment more concrete, what you actually commit yourself to'. Both the financier, i.e. the regional sales manager, and the end user said that the EUD prototype made it easy to identify the parts of the functionality they could expect to be implemented during specific iterations.

In partial support of mobilisation for design action. During the third and fifth projects the systems developers and project managers had initial problems with release and iteration planning. The main reason relates to the complexity and quality of the technical implementation of the two EUD prototypes. As has been discussed above, the systems developers found it difficult to divide the functionality into self-contained parts that they could use as user stories and later divide into work tasks, since there were many interdependencies in the EUD proto-

Published by AIS Electronic Library (AISeL), 2013

type. As a result, it took longer to stabilise plans, and thus the political ecology, than was initially expected by the two social worlds.

6.4 Legitimise design knowledge

In support of legitimising design action. From the political point of view, agreements on requirements are important to legitimise design knowledge, which in our case meant the requirements expressed through the EUD prototypes. Such agreements were found during all investigated cases. One example of an important agreement during the first project was the decision to go ahead with the final information system. This decision was made during a gate meeting where the EUD prototype was used as a demonstrator. At the meeting, several options were identified: (a) to go ahead with the complete project, (b) to go ahead with parts of the project, or (c) not to go ahead at all. The regional sales manager made the decision to go ahead with the project, based on the opinions of the end user group (employees at the regional sales office that had tried the EUD prototype) and the project leader, who participated in the meeting. The presence of the regional sales manager meant that this decision could include funding for the development of the final information system. It also meant that the design in the EUD prototype was not officially challenged.

Despite these high-level decisions, which increased the political strength of the design decisions found in the EUD prototype, the systems developers still appeared to distance themselves somewhat from the functional design. By way of example, one of the systems developers involved in the first project said: 'I do not know if it [the design] is correct'. However, our analysis does not show that the systems developers explicitly questioned the requirements or design decisions from a business point of view.

7 Discussion

The communication of requirements specifications has remained a perennial problem in systems development over the years. In this study we followed six systems development projects that used EUD as a requirements engineering technique. We analysed the functional and political ecologies of these projects using the concept of design boundary object as a theoretical lens. Table 4 shows the patterns that we found. The leftmost column shows the case study number, while the remaining columns show the four features of design boundary objects and the results found with regard to the functional and the political ecologies. We used + signs to illustrate support of the feature, (+) signs to illustrate partial support of the feature, and, finally, – signs to illustrate lack of support of the feature.

Table 4 reveals that the EUD prototype supported a shared functional representation, as long as the projects did not grow large in size. At the same time, the end users and the systems developers did not have a shared political view of the requirements in any of the cases. With regard to functional ecology, our analysis shows that the systems developers were able to transform design knowledge; in other words, they found refined solutions for the final information

system, based on the EUD prototype. Politically, however, they were not given control of all the requirements. Requirements that were based on business rationale remained with the end user. When analysing the mobilisation for design action we found that in the functional ecology the EUD prototypes provided sufficient information for implementing a refined solution. Primarily, the implemented refinements were technical. However, these were the requirements handed over to the systems developers. From the viewpoint of political ecology, the EUD prototype supported the mobilisation of resources among the systems developers. Table 4 also shows that when projects had difficulties in establishing a shared functional representation of the EUD prototype, they also had initial problems with release and iteration plans, i.e. the mobilisation of resources. Finally, the EUD prototypes contributed to legitimising design knowledge in both the functional and political ecologies.

Case no	*Design boundary object features*							
	Shared Representation		*Transforming design knowledge*		*Mobilising design action*		*Legitimising design knowledge*	
	Functional ecology	*Political ecology*	*Functional ecology*	*Political ecology*	*Functional ecology*	*Political ecology*	*Functional ecology*	*Political ecology*
1	+	–	+	(+)	+	+	+	+
2	+	–	+	(+)	+	+	+	+
3	(+)	–	+	(+)	+	(+)	+	+
4	+	–	+	(+)	+	+	+	+
5	(+)	–	+	(+)	–	(+)	+	+
6	+	(+)	+	(+)	+	+	+	+

Table 4. Patterns found in the six cases with regard to supporting the four features of design boundary objects and the functional and political ecologies (+ in support of, (+) partial in support of, - no support of)

Based on these findings, some notable lessons can be learned with regard to: a) the advantages and disadvantages of using EUD as a requirement engineering technique, b) EUD and requirements engineering research, and c) the analytical framework of design boundary objects.

Published by AIS Electronic Library (AISeL), 2013

7.1 Advantages and disadvantages of using EUD as a requirement engineering and design technique

Our findings show that EUD works well as a requirements engineering technique when considering the functional ecology of a design boundary object. Consequently, EUD prototypes can be used as vehicles for exchanging knowledge between end users and systems developers. EUD was found to provide stable requirements based on extensive testing carried out by end users. The systems developers used these requirements to create refined solutions when implementing the final information systems.

We found few instances of political shared representation; in other words, where the systems developers could account for the requirements, i.e. design decisions on what to build, from a business point of view. From a practical point of view this finding can be seen as both advantageous and disadvantageous to the project. It is advantageous for end users to be able to contribute to a solution without the systems developers having to completely understand the design rationale; in this case, design decisions are based on the end users' business knowledge. In practice, this can be useful when developing information systems in areas of high business complexity where it is difficult to map the know-how of the end users. Hence, EUD can be viewed as an alternative method for systems developers who are considering the use of such requirements engineering techniques as task analysis, observations or ethnography (Maiden and Rugg, 1996) for this purpose.

The major disadvantage of the technique is the systems developers' lack of familiarity with the business design rationale. The systems developers tend only to improve the technical side of the developed information systems, because this is the part of the project over which they have control and power. Hence, when using EUD as a requirements technique one should not expect systems developers to act as catalysts to improve the business logic. Instead, we believe that this technique should be used on well-demarcated areas, where the intention is to take advantage of the end users' extensive know-how. Another reason for using this technique on well-demarcated areas is that the size of the EUD prototypes seems to negatively affect the shared functional representation, i.e. more resources are needed to reengineer the EUD prototype into requirements that can be used by the systems developers.

7.2 EUD and requirements engineering research

Existing research on EUD claims that the designs of end user developed artefacts are volatile (Costabile et al. 2006) and that the design quality is questionable (e.g., Ditlea 1987; Teo and Tan 1999; Panko and Halverson 2001; McGill and Klobas 2005; Karlsson 2008). When using these artefacts as requirements engineering specifications it suggests problems in: a) the functional ecology with regard to transforming design knowledge and to mobilise for design actions, and b) the political ecology to mobilise design actions. However, our results show that the systems developers did not encounter any situations where they had to redesign the final information systems as a result of changing requirements. Indeed, they stated that the end users made few changes to the requirements. The reason seems to be that, functionally, design knowledge was legitimised by the end users through extensive testing.

Likewise, the systems developers experienced few difficulties in planning resource allocation. This may be due to the small size of all six studied projects. Resource planning is inevitable harder in large projects, where a larger number of systems developers are involved. But given the overall size of the projects, we can conclude that the business complexity of the projects did not affect the resource planning. However, the quality of the implemented EUD prototype affected the resource allocation process. It took longer to stabilise plans when the designs of the EUD prototypes were unstructured. This is hardly surprising since the systems developers had to spend more time reengineering existing functionality.

In their early research, Dodd and Carr (1994) showed the combination of EUD and systems development as a theoretical concept in their discussion of end user-led systems development. More recently, Floyd et al. (2007) showed the potential in using end user developed information systems as sketches of what to build. Our six cases show that this is not only a theoretical concept; companies actually use this combination from time to time. Hence, this research extends on existing requirements engineering research, by adding EUD to evaluations of requirements engineering techniques' effectiveness put forward by Maiden and Rugg (1996) and Malcom (2001) in order to address the tacit dimension of knowing (Polanyi 1983). Our study builds on the work of Pitts and Browne (2007), who put forward a call for recommendations regarding the effectiveness of investigative techniques.

As discussed in the previous section, our findings are not clear-cut. In one respect, EUD seems to be an efficient requirements engineering technique, where systems developers refine the solutions from a technical and usability point of view. These findings corroborate earlier results given by Dodd and Carr (1994), who discussed that systems developers add other elements to the solutions, such as security. However, the systems development results found in the functional ecology occur without shared politically representation between the social worlds, i.e. there is a tacit dimension to the requirements that the EUD prototypes represent. In another words, requirements are passed on 'blindly' for implementation without the systems developers understanding why they are implementing certain designs. This a major difference from other techniques, such as observation and ethnography, where the systems developers try to learn more about the end user's craftsmanship and business logic (Maiden and Rugg 1996). As a result, systems developers did not take control and power of all requirements and design decisions, and did not act as a catalyst to leverage business logic. Consequently, this is in line with Davis' (1988) criticism of EUD as a concept, that systems developers' capacity as facilitators is not used to its full potential. It means that EUD is not an effective requirements engineering technique if it does not serve as a starting point for further discussions between the two social worlds.

Consequently, our results show that there is an important political aspect when crossing EUD and systems development, which has not been shown before. Our research, therefore, extends findings by Rittenberg and Senn (1993), and Floyd et al. (2007). They only briefly addressed the receiving social world, i.e. the systems developers and they way they act and are allowed to act within the political ecology. Their main focus was on how the end users acquire more powerful tools and become more knowledgeable. Our findings show that this comes at a price when considering the overall systems development process: the systems developers feel a lesser degree of ownership of the design and become less knowledgeable. It means that introducing EUD to systems development affects the balance of power in the political ecology, where systems developers mainly take control over technical requirements and leave the business logic to

Published by AIS Electronic Library (AISeL), 2013

the end users, instead of working as facilitators for all types of requirements. Hence, our results show that there is a trade-off between the end users' empowerment and the systems developers acting as facilitators. What kind of long-term effects this shift in power has on, for example, maintenance has not been possible to investigate in our study.

7.3 The analytical framework of design boundary objects

This research contributes with an elaboration of the design boundary object as a theoretical lens. The framework presented by Bergman et al. (2007) is not an operationalisation targeting analysis of requirements techniques per se. Rather, it contains a general description of the features of design boundary objects and the two ecologies. We have developed a set of questions to be used when analysing requirements techniques (see the rightmost column of table 3). They can also be used to analyse other requirements techniques, making it possible to conduct more exhaustive comparisons of how different techniques affect the functional and the political ecologies of projects.

7.4 The limitations of the research

In this paper, we report on projects that were limited, both in terms of number of man-hours and project members. Thus, all the investigated projects were small and set within the context of industrial systems development. Both the end users and the systems developers were treated as homogeneous social worlds, which may not be the case in larger projects. In larger projects, both end users and systems developers may be specialised in different areas, creating a more complex web of social worlds.

Our analysis showed that systems developers did not take control and power over all requirements when EUD was used as a requirements engineering technique. We can conclude that design decisions that concern business logic remained with the end users. However, our data did not allow us to draw conclusions on why systems developers acted in this way during the projects, because the data was not collected with that purpose in mind.

In all the studied projects, the end users used spreadsheets as their development tool. It is by far the most commonly used computerised tool for EUD (Burnett et al. 2004); however, it may mean that our findings are restricted to problem domains associated with spreadsheets, i.e. information systems involving a high degree of calculation. In addition, all studied projects used agile systems development methods. Hence, we do not know if our findings are valid for projects using other types of tools or other types of systems development methods, such as the Rational Unified Process (Kruchten 2004).

None of the studied projects included maintenance of the developed information systems. Hence, we have not had the opportunity to investigate how the use of EUD as a requirements engineering technique affects the maintenance phase. Rittenberg and Seen (1993) have earlier pointed out that ownership issues, which are related to maintenance, are important to consider when end user developed applications are repackaged in organisations for wider distribution. It may impose new challenges when systems developers do not have control and power over all

design decisions, which means that many design decisions may not have been documented. This could mean that systems developers become more end user-dependent during maintenance, shifting the power structures even more than we have been able to show.

7.5 Identified areas for future research

Given the limitations discussed above, there is ample opportunity for future research. First, an interesting avenue for future research concerns why systems developers do not take control and power over all requirements when EUD is used as a requirements engineering technique. Such findings will give a deeper understanding of the technique's political drawbacks. Second, it is interesting to address the scalability issues and to investigate the use of EUD in larger projects. It may offer an opportunity to identify and study a multitude of social worlds and, for example, see if and how the EUD prototype can be used to settle any outstanding conflicts on requirements and designs between end user groups and/or system developer groups. Third, it is an opportunity to study projects using other tools than spreadsheets as well as non-agile systems development methods. For example, it would be interesting to study how web mash-ups, as described by Floyd et al. (2007) and Cappiello et al. (2013), could be used as design boundary objects between end users and systems developers. Especially, since this technology has become prominent during recent years. Finally, maintenance is also an important area of future research if we are to fully understand the long-term effects of using this technique during systems development.

8 Conclusions

The aim of this paper was to examine end user development as a requirements engineering technique for communicating across social worlds in systems development. For this purpose, we employed the theoretical lens of design boundary object. We found that the artefact developed by the end users can act as a vehicle for communicating their requirements. This technique has a high degree of user involvement, which takes advantage of end user know-how. As a requirement engineering technique it also has the ability to capture requirements and transfer them into the final information system without the systems developers having to completely understand the design rationale. Using this approach, the systems developers have little or no influence on business requirements. In this study, their role is mainly as technical experts, rather than as business developers, as is sometimes the case. The systems developers took control and power over technical requirements, while requirement relating to business logics remained with the end users. Consequently, the systems developers did not act as a catalyst in the systems development process, which may hamper the overall results of the information system.

Published by AIS Electronic Library (AISeL), 2013

References

Aurum, A., and Wohlin, C., (2003). The fundamental nature of requirements engineering activities as a decision-making process. *Information Software and Technology,* (45:14): 945-954.

Berg, M., and Bowker, G., (1997). The multiple bodies of the medical record: toward a sociology of an artefact. *The Sociological Quarterly*, (38:3): 513-537.

Bergman, M., Lyytinen, K. ,and Mark, G., (2007). Boundary Objects in Design: An Ecological View of Design Artifacts. *Journal of Association of Information Systems,* (8:11): 546-568.

Brancheau, J. C., and Brown, C. V., (1993). The Management of End-User Computing: Status and Directions. *ACM Computing Surveys*, (25:4): 437-481.

Burnett, M., Cook, C., and Rothermel, G., (2004). End-User Software Engineering. *Communication of the ACM*, (47:9): 53-58.

Bødker, S. (1998). Understanding Representation in Design. *Human-Computer Interaction,* (13:2): 107-125.

Cappiello, C., Matera, M., and Picoozi, M., (2013). End-user development of mobile mashups. In: *Design, User Experience, and Usability. Web, Mobile, and Product Design*, A. Marcus (ed.). Springer, Heidelberg, pp. 641-650.

Checkland, P. B., (1981). *Systems thinking, systems practice*, Wiley, Chichester.

Costabile, M. F., Fogli, D., Mussio, P., and Piccinno, A., (2006). End-user development: The software shaping workshop approach. In: *End-User Development,* H. Lieberman, F. Paterno and V. Wulf. Dordrecht (eds.), Springer, Dordrecht, pp. 183-205.

Costabile, M. F., Mussio, P., Provenza, L. P., and Piccinno, A., (2008). End Users as Unwitting Software Developers. In: *Proceedings of the 4th Workshop on End-User Software Engineering (WEUSE IV),* R. Abraham, M. Burnett and M. Shaw (eds.), ACM Digital Library, New York, pp. 6-10

Davis, G. B., (1988). The Hidden Costs of End-User Computing. *Accounting Horizons*, (2:4): 103-106.

Ditlea, S., (1987). Spreadsheets can be hazardous to your health. *Personal Computing*, (11:1): 60-69.

Dodd, J. L., and Carr, H. H., (1994). Systems Development Led by End-users: An Assessment of End-user Involvement in Information Systems Development. *Journal of Systems Management*, (45:8): 34-40.

Ehn, P., (1993). Scandinavian Design: On Participation and Skills. In: *Participatory design : principles and practices*. D. Schuler and A. Namioka (eds.), L. Erlbaum Associates, Hillsdale, pp 41-77.

Eriksson, J., and Dittrich, Y., (2007). Combining Tailoring and Evolutionary Software Development for Rapidly Changing Business Systems. *Journal of Organizational and End User Computing*, (19:2): 47-64.

Floyd, I. R., Jones, M. C., Rathi, D., and Twidale, M. B., (2007) Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software. In: *Proceedings of the 40th Hawaii International Conference on Systems Sciences*, R. H. Sprague Jr. (ed.), IEEE Computer Society, Los Alamitos, pp: 86-96.

Goldkuhl, G., (2012). Pragmatism vs interpretivism in qualitative information systems research. *European Journal of Information Systems*, (21:2): 135-146.

Greenbaum, J., and Kyng M., (1991). *Design at work: cooperative design of computer systems*. Lawrence Earlbaum, Hillsdale.

Hansen, S., Berente, N., and Lyytinen, K., (2008). Emerging principles for requirements processes in organizational contexts. *Networking and Information Systems*, (13:1): 9-35.

Henderson, K., (1991). Flexible Skecthes and Inflexible Data Bases: Visual Communication, Consription Devices, and Boundary Objects in Design Engineering. *Science, Technology, & Human Values*, (16:4): 448-473.

Karlsson, F., (2008). Using Two Heads in Practice. In: *Proceedings of the 4th Workshop on End-User Software Engineering (WEUSE IV),* R. Abraham, M. Burnett and M. Shaw (eds.), ACM Digital Library, New York, pp. 43-47

Karten, N., (1990). Curing users tunnel vision. *Computerworld* (24:18): 116

Kensing, F., and Munk-Madsen A., (1993). PD: Structure in the toolbox. *Communication of the ACM*, (36:4): 78-85.

Klein, H. K., and Myers, M. D., (1999). A set of principles for conducting and evaluating interpretative field studies in information system. *MIS Quarterly*, (23:1): 67 - 94.

Ko, A. J., Abraham R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Lawrence, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Scaffidi, C., Shaw, M., and Wiedenbeck, S., (2011). The state of the art in end-user software engineering. *ACM Computing Surveys*, (43:3): 21:21-21:44.

Kreie, J., Cronan, T. P., Pendly, J., and Renwick, J. S., (2000). Applications development by end-users: can quality be improved? *Decision Support Systems*, (29:2): 143-152.

Kruchten, P., (2004). *The rational unified process: an introduction*, Addison-Wesley, Reading, MA.

Maiden, N. A. M., and Rugg, G., (1996). ACRE: selecting methods for requirements acquisition. *Software Engineering Journal*, (11:3): 183-192.

Malcolm, E., (2001). Requirements acquisition for rapid applications development. *Information & Management*, (39:2): 101-107.

Martin, J., (1991). *Rapid application development*, Macmillan, New York.

McDaid, K., Rust, A., and Bishop, B., (2008). Test-Driven Development: Can it Work for Spreadsheets? In: *Proceedings of the 4th Workshop on End-User Software Engineering (WEUSE IV),* R. Abraham, M. Burnett and M. Shaw (eds.), ACM Digital Library, New York, pp. 25-29.

McDermid, J. A., (1994). Requirements analysis: Orthodoxy, fundamentalism and heresy. In: *Requirements engineering: social and technical issues*, M. Jirotha, and J. A. Goguen (eds.) Academic Press, London, pp. 17-40.

McGill, T., (2004). The Effect of End User Development on End User Success. *Journal of Organizational and End User Computing*, (16:1): 41-58.

McGill, T. J., and Klobas, J. E., (2005). The role of spreadsheet knowledge in user-developerd application success. *Decision Support Systems*, (39:3): 355 - 369.

Mogensen, P., (1992). Towards a provotyping approach in systems development. *Scandinavian Journal of Information Systems*, (4): 31-53.

Moore, J. E., (2002). The focus of research in end user computing: where have we come since the 1980s? *Journal of End User Computing*, (13:1): 3-22.

Mumford, E., (1981). Participative Systems Design: Structure and Method. *Systems, Objectives, Solutions*, (1:1): 5-19.

Ocker, R., Hiltz, S. R., Turoff, M., and Fjermestad J., (1996). The Effects of Distributed Group Support and Process Structuring on Software Requirements Development Teams: Results on Creativity and Quality. *Journal of Management Information Systems*, (12:3): 127-153.

Panko, R. R., (1998). What We Know About Spreadsheet Errors. *Journal of End User Computing*, (10:2): 15-21.

Panko, R. R., and Halverson R. P., (2001). An Experiment In Collaborative Development To Reduce Spreadsheet Errors. *Journal of the Association of Information Systems*, (2:1): 1-31.

Patton, M. Q., (1990). *Qualitative evaluation and research methods*, Sage, Newbury Park.

Pitts, M. G., and Browne, G. J., (2007). Improving requirements elicitation: an empirical investigation of procedural prompts. *Information Systems Journal*, (17:1): 89-110.

Polanyi, M., (1983). *The tacit dimension*, Peter Smith, Gloucester.

Powell, S. G., and Baker, K. R., (2003). *The Art of Modeling with Spreadsheets: Management Science, Spreadsheet Engineering, and Modeling Craft*, John Wiley & Sons, Hoboken.

Rittel, H., (1984). Second Generation Design Methods. In: *Developments in Design Methodology*, N. Cross (ed.), John Wiley & Sons, Chichester, pp. 317-327.

Rittenberg, L. E., and Senn, A., (1993). End-user computing. *The Intenal Auditor*, (50:1): 35-40.

Sayles, J., (1990). Beware the dark side of 4GLs. *Computerworld*, (24:35): 100.

Schwaber, K., and Beedle, M., (2001). *Agile Software Development with Scrum*, Prentice Hall, Upper Saddle River.

Star, S. L., (1993). Cooperation without consensus in scientific problem solving: dynamics of closure in open systems. In: *CSCW: Cooperation or Conflict?*, S. M. Easterbrook (ed.), Springer Verlag, London, pp. 93-106.

Star, S. L., and Griesemer, J. R., (1989). Institutional Ecology, 'Tranlsations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, (19:3): 387-420.

Taylor, M. J., Moynihan, E. P., and Wood-Harper, A T., (1998). End-user computing and information systems methodologies. *Information Systems Journal*, (8:1): 85-96.

Teo, T. S. H., and Tan M., (1999). Spreadsheet development and 'what-if' analysis: quantitative versus qualitative errors. *Accounting Management and Information Technologies*, (9:3): 141-160.

Walsham, G., (1995). Interpretive case studies in IS research: nature and method. *European Journal of Information Systems*, (4) 74-81.

Yin, R. K., (1994). *Case study research: design and methods*, SAGE, Thousand Oaks.