Pergamon

Information and Organization 12 (2002) 249–281

INFORMATION AND ORGANIZATION

www.elsevier.com/locate/infoandorg

Towards an understanding of the use of problem and design spaces during object-oriented system development

Sandeep Purao [a,d*], Matti Rossi [b], Ashley Bush [c]

[a] *Department of Computer Information Systems, J. Mack Robinson College Of Business, Georgia State University, Atlanta, GA 30302, USA*

[b] *Helsinki School of Economics, P.O. Box 1210, FIN-00101, Helsinki, Finland*

[c] *Department of Management Information Systems, College of Business, Florida State University, Tallahassee, FL 32306, USA*

[d] *School of Information Sciences and Technology, Penn State University, University Park, PA 16801, USA*

Received 1 June 2001; accepted 1 March 2002

Abstract

The importance of, and distinction between, problem and design spaces has been recognized in studies of information systems development (ISD). With increasing acceptance of object-oriented techniques, which promise close mirroring of real-world concepts in the IS artifacts, this distinction becomes even more important. In spite of a rich literature stream dealing with the general notion of 'design,' several inadequacies remain in our understanding of the ISD processes—one of these is the recognition and impact of problem and design spaces. In this paper, we analyze processes followed by two developers engaged in a non-trivial development task using the object-oriented modeling techniques—with a view to understanding their use of, and explorations in, problem and design spaces. Our analysis provides evidence for a distinction between the two spaces, and interprets the developers' engagements and behaviors to structure the two spaces with the help of object-oriented modeling techniques. Several interesting findings emerge from our analysis, including the overlapping of spaces due to the use of object-orientation, disturbing patterns such as design fixation, interplay between simulation, expansion and validation in the design space, and the varying use of modeling techniques to structure the two spaces. Our analysis, supported by prior literature, provides a grounded description of some phenomena that have, hitherto, had only intuitive or prescriptive support. Based on these, we argue for more methodological and tool support for explorations of problem

* Corresponding author. Tel.: +1-404-651-3859; fax: +1-404-651-3842.
E-mail address: spurao@gsu.edu (S. Purao).

1471-7727/02/$ - see front matter
PII: S1471-7727(02)00006-4

space, and enhancements to current approaches, for better integration of problem and design spaces. © 2002 Published by Elsevier Science Ltd.

1. Introduction

The information system development (ISD) process requires the developer to understand the requirements of the users in a given domain and translate these requirements, using a representation technique, into a model that may be implemented as an information system (IS) artifact. The ideas of 'Problem' and 'Design' spaces therefore, represent a key dimension that underlies ISD processes. Though the separation of the two spaces has rarely been emphasized in prior research on ISD processes, the theme has persisted in writings on ISD (e.g. Guindon, 1990; Guindon, Krasner, & Curtis, 1986; Simon, 1996). Mathiassen & Munk-Madsen, (2000), for instance, describe the concept of the problem domain, alluding to the importance of appropriate mapping between problem and design space. Guindon (1990 p. 285–86) identifies the problem domain as a subset of the real world with which a computer-based information system is concerned, but not the design solution describing the system itself. The sub-fields of requirements analysis (Pohl, 1994), and system modeling (Rossi & Siav, 2001) have been acknowledged as important. However, the interplay between these two, recognizing them as problem and design space respectively, and how developers use the two spaces is a topic that is largely neglected by the prescriptive approaches. An understanding of the developers' behaviors in, and use of, the two spaces during ISD should, therefore, contribute to a better understanding of the ISD process.

Over the years, a number of studies have been carried out to better understand 'the design process' (Adelson & Soloway, 1985; Guindon, Krasner & Curtis, 1986; Sen, 1997; Vitalari, 1985). The principal aim of this research stream has been to provide an account of the underlying structures associated with the rather private actions of developing IS artifacts. The studies are concerned with interpreting the interior situational logic, decision-making and related activities of IS developers. Unlike prescriptive methodologies (Booch, 1994; Booch, Jacobson, & Rumbaugh, 2001) that represent *simplified* accounts of the intricate and esoteric act of 'designing,' and pay scant attention to the developers' micro-processes, the focus of this research stream is building an understanding of the *actual* behaviors and actions of individual system developers. In this study, our specific focus is the developers' use of, and explorations in, problem and design spaces. Given the nature of inquiry, we use a constructivist approach to data collection and analysis (Guba & Lincoln, 1989). Specifically, we follow the grounded theory development approach suggested by Pare' and Elam (1997).

While designing the study, we have attempted to overcome three shortcomings observed in the earlier studies. First, unlike earlier studies (Adelson & Soloway, 1985; Guindon, Krasner & Curtis, 1986; Sen, 1997), which investigate individual

developers working on simple tasks, in single sessions[1], this study examines the developer as she tackles a non-trivial task through multiple sessions. Second, instead of devising a classification scheme anew for codifying developer behaviors, this study builds on categories provided by earlier researchers, attempting to contribute in the narrowly defined aspect of interest: problem and design spaces. Third, unlike prior studies, this study includes analysis of modeling technique usage. It can be argued that the modeling technique is probably more important in ISD than in other professions (such as architecture or engineering), because the IS artifact rarely bears direct correspondence to a physical realization. With the acceptance of object-orientation (Booch, Jacobson, & Rumbaugh, 2001), which has added further abstraction to the modeling process, this importance has increased even further. The objective of this paper, therefore, is *to investigate the use and exploration of problem and design spaces by IS developers—engaged in tackling a non-trivial object-oriented system development task over multiple sessions.*

In the information systems field, due to the constantly changing landscape of technology, several fundamental questions have remained unanswered. A theory-base that squarely addresses the core activity of ISD has been slow in emerging (Lyytinen, 1987). With the accelerating pace of change, researchers' focus has increasingly shifted to new concerns. Unlike other professional disciplines concerned with design (e.g. mechanical engineering (Visser, 1990), architecture (Goldschmidt, 1997)), which have devoted considerable research to observing the actual processes of designers, the focus in Information Systems has been mainly on prescriptive approaches. By addressing the proposed question, this research hopes to contribute to theory development indigenous to the Information Systems field. At a pragmatic level, understanding of the developers' behaviors in and use of the two spaces during the development process has implications for methodological support needed for ISD processes in these spaces.

The paper is organized as follows. In section 2, we introduce and amplify the ideas of problem and design spaces, highlight essential elements of O-O development, and review prior studies of individual developers. In section 3, we describe the methodology employed for the multi-session study, the data collection approach and specific tasks undertaken during data analysis. Section 4 presents our findings, analysis and interpretations grouped under multiple subsections. In section 5, we discuss contributions of our findings to the ISD research stream and implications for practice.

2. Prior research

2.1. Problem vs design spaces

The idea of problem and design spaces is important in artifact design processes. The problem space can be described as the metaphoric space that contains mental

[1] The locus of investigation has shifted to teams or organizations (Curtis, Krasner & Iscoe, 1988; Tan, 1994) for investigation of larger, more complex tasks.

representations of the developer's interpretation of the user requirements. The design space, on the other hand, is the metaphoric space that contains mental representations of the developer's specific solutions, based on which she creates models and specifications for building the IS artifact. Newell and Simon (1972) first used the term 'problem space,' defining it as the metaphoric space wherein the problem solver moves from an initial (problem) state through intermediate states to a goal (solution) state by applying appropriate operators (Simon, 1996). The former can be considered 'interpretive' as opposed to the latter, which is 'constructive'. Their view places the problem and the solution on the same plane. This is contrary to what we may intuit for ISD processes, where the development of a 'solution' i.e. a design, proceeds through proposals, expansions and challenges. These activities clearly represent moves within a design space and continue after an 'initial' goal state is reached—suggesting the existence of a *separate* design space.

Interpretations that allude to existence of design space abound as well. Rowe (1987) describes generative processes, or operations, that allow the designer to take knowledge states as input, or as starting positions, and produce new knowledge states as outputs. The representation of new, but incomplete states may resemble partial design solutions suggesting the existence of a separate design space. Gero and McNeil (1997 p.23) observe designer navigation between different levels of problem abstractions and as different design strategies used by the designer. Oxman (1997, p. 336), while interpreting behaviors of designers who are given a partial solution and asked to adjust it, comments that they "…could easily explore different solution spaces....". Guindon (1990 p. 285–86) clearly identifies the problem domain as a subset of the real world with which a computer system is concerned but not the design solution describing the computer system itself.

Mathiassen and Munk-Madsen (2000) describe a problem domain as 'that part of the context that is administrated, monitored or controlled by a system. They describe an application domain as 'the organization that administrates, monitors or controls a problem domain'. The application domain, therefore, includes not only the context in which the problem domain is interpreted (administrated, monitored or controlled), but also includes the milieu in which the 'information system' to be designed, will be inserted. In this view, systems contain models of problem domain, and their critical quality is the goodness of the fit, because the future problem domain will be administered through this model. There is, thus, a constant interplay between the understanding of the current state of the affairs (problem space) and the design of the possible future states of systems (design space).

Other manifestations of this distinction are implicit in prescriptive writings. Yourdon (1995), for example, indicates that developers should strive to build 'essential' models that are uncluttered with technological choices. A similar distinction is made more recently by Larmon (1998), who argues for making conscious choices between 'real' and 'essential' versions of use cases. A contrary set of terms betrays the framing from earlier writings on prescriptive ISD. Gane and Sarson (1979) identify physical (representing the current work practice, that is, the problem space) as opposed to logical (representing technology-independent solutions, that is, design space) models. These distinctions suggest that methodologists recognize the need for different

decisions dealing with problem vs design space. The two spaces, thus, clearly represent a dimension, orthogonal to the ISD processes, that has been an implicit, enduring theme over the years.

2.2. Object-orientation

The object-oriented (O-O) development paradigm is based on a message-object view of IS, instead of the traditional operator-operand model (Cox, 1984). Our discussion[2] here is focused on key aspects of object-orientation that can impact the ISD process, specifically, the use of problem and design spaces.

The first important consequence of using O-O is that elements in the information system must be defined as objects that can interact with each other by sending messages (Wegner, 1997). This view is closer to the way human beings perceive and interact with real-world entities. Unlike the traditional, operator-operand view, the O-O perspective does not require translation of the understanding of a real-world phenomenon into an algorithmic view. As a result, developers can continue to use the users' vocabulary during the ISD process, blurring the distinction between concepts in the problem space and those in the design space. For example, the word 'Customer' may be used to refer to the real customer (in the problem space) or its representation (in the design space) by the class 'Customer.' Clearly, a potential consequence is the overlap between concepts across problem and design spaces. A related aspect of O-O modeling techniques is the notion of use cases (Jacobson, Christerson, Jonsson, & Overgaard, 1992), which focuses on the identification and specification of typical scenarios that expressly deal with the context in which the system will be inserted—affording the problem space an important place in the development process.

The second important feature of O-O is the multiple views it provides for exploring the design space. Like architectural drawings, blueprints and cross-sections, the object-oriented representation techniques involve creation of multiple views such as use cases and class diagrams. For example, the Unified Modeling Language suggests at least three complementary perspectives for specifying the information system, structural, functional and dynamic (Booch et al., 2001, D'Souza & Wills, 1999 p. 45).[3] Using multiple techniques allows checking for internal consistency of the specification and designing of a more complete specification of the solution.

Finally, the representational modes for object-oriented design require neither a top-down nor a bottom-up approach to problem-space planning and design procedures (Rowe, 1987 p. 67–71). Instead, an iterative-incremental approach (Booch, 1994) is considered more appropriate, which suggests growing and refining the artifact by taking into account additional scenarios, and/or considering progressively detailed accounts of given scenarios. The advocated development process with use

[2] Excellent descriptions of important concepts in the paradigm are available elsewhere (Booch, 1994; Booch, Jacobson, & Rumbaugh, 2001; Jacobson, Christerson, Jonsson & Overgaard, 1992).

[3] UML (Booch, Jacobson, & Rumbaugh, 2001) terms these (a) class diagrams, (b) use cases and (c) interaction diagrams respectively (and suggest others to supplement this set).

cases (Kruchten, 1998) involves incremental refinement and specification of the use cases. A less-specified scenario requires a higher relative degree of focus on the problem space, whereas a highly-specified scenario requires a move towards the design space, with the iterations providing multiple opportunities for crossing across problem and design spaces. The picture that emerges from the above suggests that both problem and design spaces are important for ISD processes that adopt the O-O paradigm.

2.3. Individual development processes

ISD involves a process in which the developer progresses from a description of requirements to a model of an IS artifact that will satisfy these requirements. It begins with a problem where the actions necessary to obtain a solution are not obvious. The actions that follow may, in turn, focus on solving this problem as well as additional problems that are brought up through the definition and redefinition of the problem (Rowe, 1987 p.39). Thus ISD can be defined as an iterative process of solving wicked problems (Buchanan, 1992)[4].

One major obstacle to studying and understanding the development process is the 'messiness' of the task. Developers rely on a variety of techniques—often simultaneously—to perform and manage the process. To address this gap, descriptive studies have been conducted in various professional disciplines (mechanical design, architecture, IS development). Two recent studies of architectural design represent the nature of research and interpretations obtained. First, Gero and McNeill (1997) develop a multi-dimensional model to describe these activities. These include different levels of design foci such as behavior or structure and different micro and macro design strategies. In another study, Goldschmidt (1997) uses the 'reflective practice' framing provided by Schön (1983) to interpret different design tasks.

Few studies of ISD have focused on the individual developer. An early study by Guindon, Krasner and Curtis (1986) focuses on breakdowns observed during early ISD activities. Based on a study of three developers engaged in creating a solution for an N-lift elevator problem, they identify a number of breakdowns in two broad categories—knowledge-related and those due to cognitive limitations. Another study by Adelson & Soloway (1985, 1988) provides a model of behaviors of both expert and novice designers in domains in which they had varying levels of familiarity. They identify six major activities: model formation, expansion, simulation, constraint representation, note making, and label retrieval. In another study, Guindon (1990) investigates behaviors of three software developers for a similar N-lift elevator problem with a view to identifying opportunism during the development process. She concludes in favor of opportunistic, instead of top-down, decomposition during the early stages of design. Sen (1997) confirms the role of opportunism in the software

[4] A 'wicked' problem (Buchanan, 1992) (a) defies the possibility of a definitive formulation, (b) exhibits no stopping rules for the development activity, (c) allows differing formulations to imply different solutions, and (d) allows alternative solutions that have only degrees of acceptance, unlike 'ill-defined' problems that *eventually* lend themselves to definition of criteria for judging the solutions.

design process by proposing a cognitive model for examining demand-side reuse tasks during ISD. He identifies a number of reuse mechanisms employed by the developers and concludes that opportunism does exist in the reuse process. Table 1 below summarizes a representative sample of studies of individual developers, from different disciplines. Detailed accounts of earlier studies are also available in Dorst (1997) and Cross (1999).

From the table, it is clear that use of 'think aloud' protocols continues to be the dominant mode of data collection with a few studies in architecture beginning to use results from prior studies for data analysis (e.g., Goldschmidt, 1997). The dominant type of task investigated, however, remains a simple one, requiring a single design session, particularly in the IS design studies (for example, Guindon et al., 1986; Sen, 1997). Finally, the analysis of protocols appears to ignore the underlying representation mechanism such as the object-oriented modeling techniques. Against the backdrop of the above literature review, our study represents a first effort that combines multi-session analysis of a complex task, with explicit recognition of a modeling technique, with a view to building our understanding of a narrowly defined aspect of ISD: the use of problem and design spaces.

3. Methodology

The research method followed for this study is case-study-based grounded theory development (Eisenhardt, 1989; Yin, 1984). Following the general roadmap outlined by Pare' and Elam (1998), we gather and analyze data from multiple developers,

Table 1
A selective survey of prior research on design behaviors[a]

Researcher(s)	Study Design		Framing	Key Contribution
Adelson and Soloway, 1985	I	Single-session, Five Designers	Domain experience and designer expertise	Model of design behaviors
Gero and McNeil, 1997	A, M	Single session, Three designers	Strategies during the design process	Coding scheme and method for the analysis of design protocols
Sen, 1997	I	Single session, Multiple designers	Demand side cognitive tasks during reuse	Establishing the role of opportunism in reuse
Goldschmidt, 1997	M	Single session, One designer	Indeterminism in the design process	Modified model of design problem space
Guindon et al., 1986	I	Single session, Three designers	Obstacles to design	Identification of breakdown categories
Visser, 1990	M	Multiple sessions, One designer	Blackboard model	Opportunistic element of design

[a] Key: I=Information System Design, A=Architectural Design, M=Mechanical Design.

engaged in tackling a non-trivial object-oriented design task. Table 2 shows the overall approach. The rightmost column contains instantiations of each phase for this research.

Our intent is to develop a grounded theory of the existence, use and exploration of Problem and Design spaces during object-oriented system development. The grounded theory method (Glaser, 1992; Strauss & Corbin, 1990) is a 'qualitative research method that uses a systematic set of procedures to develop an inductively derived theory about a phenomenon' (Strauss and Corbin, 1990, p. 24). It involves a general approach to analysis that does not depend upon particular disciplinary perspectives, that is, it is appropriate for the multidisciplinary nature of research in information systems. The benefit of the grounded theory approach is that the resulting theory is intimately tied to the evidence (Eisenhardt, 1989).

Table 2
A case-study-based methodology [adapted from Pare' and Elam (1998)]

Phase: Preparation		
Getting started	Identification of research questions	*How do developers use Problem and Design spaces during object-oriented information systems development?*
	Use of a priori constructs, No theory or hypotheses specified	Adelson and Soloway (1985) for initial coding, No a priori theory or hypotheses specified
Phase: Design		
Case selection	Specified population, Theoretical sampling	Participants from the potential and actual developers of information systems
Data collection	Qualitative Data Collection	Use of verbal protocols as data—from two developers over three sessions
Phase: Coding		
Data coding	Coding as a precursor to analysis	Extension of existing set of constructs, Identification of the use of modeling techniques
Phase: Analysis		
Data analysis	Within-case analysis, Cross-case patterns identification	Preparation of visual displays to aid in analysis, Search for patterns aided by interpretations
Shaping hypotheses	Search for evidence across cases, Search evidence for 'why' behind relationships	Confirmatory evidence literal replication strategy, Use of technical literature on system development to understand 'why'
Enfolding literature	Comparison with similar literature, Comparison with conflicting literature	Comparison with research on information systems development

A requirement of the grounded theory is that the researchers demonstrate theoretical sensitivity (Glaser, 1992) by being well-grounded in technical literature, from personal and professional experience and in the collection and analysis of data (Strauss & Corbin, 1990). Two of the researchers involved in this study are primarily engaged in research on several technical areas of ISD, specifically focusing on object-oriented techniques, and the unified modeling language. Use of the grounded theory approach is, therefore, an attractive choice as the core methodology. The interplay between emergent theory, based on evidence reported in this paper and the technical literature dealing with the use of object-oriented techniques for developing information systems is particularly relevant in later phases of analysis.

3.1. Setting

The research setting engaged the developers in multiple sessions performing a single task—developing an Elevator Control System. The description of requirements (Yourdon & Argila, 1996) covered several aspects of an elevator system. Compared to the N-lift problem statement (Guindon, 1990; Guindon et al., 1986), our problem statement was much larger, less specified and allowed different interpretations. It was aided by a short list of important events leading to identification of use cases such as: 'an elevator is called from a floor.' These were expected to provide the developers a number of starting points. The deliverables required were a statement of scope, object-oriented models such as a structural (class) model, dynamic (interaction) models, and decisions about the system architecture.

The developers were drawn from a graduate level course in object-oriented system development as part of a masters degree program in Information Systems. The use of students as developers is comparable to other recent studies such as Sen (1997) and Adelson and Soloway (1988). One issue in protocol studies is generalizability of the findings since the number of participants is very small. There is simply no reliable way, given the current maturity of the field, to decide how representative our participants are of the larger population of developers (Guindon, 1990). There is no standard type of individual who becomes a software developer. They can differ in education, experience and attitude. The participants we selected had some prior experience in the IS industry and were obtaining a specialized master's degree in IS while continuing work. They were, thus, representative of at least some segment of the population of actual developers. For selecting cases for building grounded theory, several possibilities have been identified, including filling-in of theoretical categories, providing examples of polar types or literal replication (Eisenhardt, 1989; Benbasat et al., 1987; Pare', 1995). Given the nature of our research, we adopted a literal replication strategy, where similar, not contrasting, results were predicted from each participant. The number of replications is a matter of discretionary and judgmental choice (Yin, 1984; Eisenhardt, 1989). Given the intensity of analysis, the focus of the study, and prior work dealing with study of information system development (which uses one to three subjects, with the exception of Adelson and Soloway (1985), who used five in order to derive a set of behaviors we use in this research), two to three participants were considered sufficient.

Table 3
Duration of sessions for the two participants

Subject	Session 1	Session 2	Session 3
Andrea	45 minutes	86 minutes	38 minutes
Stanley	40 minutes	67 minutes	40 minutes

During the development process, the developers were given the problem statement, a reference book (Fowler, 1999), a copy of the UML Notation Version 1.1 (Booch, Jacobson, & Rumbaugh, 2001) and reusable analysis patterns (Coad, 1995). Each developer was allowed to work over three sessions (see Table 3). The sessions were spread over about a week, scheduled at the participants' convenience. The sessions were conducted in a room, occupied solely by the developer, who was asked to *think aloud* (Ericsson & Simon, 1993) as she proceeded through the design task. A tape recorder captured the verbal protocols. Of the three developers who participated in the study, two (identities concealed) provided complete verbal protocols. The third developer remained silent for long periods during the study. Few verbal protocols were, therefore, captured for this participant. This provided two sets of protocols, from Andrea and Stanley (names concealed).

3.2. Coding

For coding the transcribed verbal protocols, the model of behaviors (see Table 4) suggested by Adelson and Soloway (1985) was used as the starting point. The choice of the model was based on a number of factors, both theoretically rigorous and pragmatic. First, this model provided a well-accepted and often-cited model of developer behaviors that has been shown to be useful for describing both novices as well as experts. This ensured possible comparisons with other studies and a feasible path to later replication. Second, their model represents one of the few models specifically aimed at ISD and *prima facie*, covers the entire development process including problem definition (constraint representation), solution generation (model formation,

Table 4
Model of developer behaviors (Adelson & Soloway, 1985)

Developer Behaviors	**Explanation**
Model formation	New ideas are identified and mental models are formed.
Model expansion	The mental models are expanded to an acceptable level of detail.
Model simulation	The mental models are simulated in a dynamic manner.
Constraint representation	Implicit constraints about boundary are made explicit.
Label retrieval	Retrieving labels attached to previously solved parts to retrieve these plans.
Note making	A note is made of issues that need to be addressed later.

expansion and simulation) and strategies for planning the design process (note making and label retrieval). Table 4 shows the categories suggested by Adelson and Soloway (1985) along with a brief explanation.

As a first pass, separate and independent coding was attempted by two of the co-authors on two different sessions, followed by an assessment of agreements and disagreements among the coders. Any disagreements on the granularity of units identified as well as their classification following the categories (Table 4) were resolved by a process of negotiation and logical argumentation. As a shared frame of reference was established for application of the model, the coders restarted and conducted the coding process for the protocols jointly. This allowed immediate discussion of discrepancies and resolution of disagreements. During the coding process, the coders established the rules of coding, which included (a) identification of verbal protocols as a phrase or a sentence indicating a reasonably complete unit, and (b) classification of each unit into one or more categories. As the analysis progressed, the categories (Table 4) were refined. This resulted in the extended model underlying our coding (Table 5). The first four categories were retained without adjustments. The category 'note making' was refined to add another, 'making meta-level notes.' This represented behaviors that dealt with planning or management decisions, as opposed to recording concerns to be addressed later i.e. 'note making'. The category 'label retrieval' was refined to three sub-categories—'retrieving domain knowledge', 'retrieving technique knowledge' and 'retrieving from experience base'—as explained in Table 5. Finally, two new categories were added—'refocusing' and 'validating'—again, explained in Table 5. A residual category, not shown in the

Table 5
Extended model of design behaviors

Developer Behaviors		Explanation
1	Concept formation	New ideas are identified and mental models are formed.
2	Concept expansion	The mental models are expanded to an acceptable level of detail.
3	Concept simulation	The mental models are simulated in a dynamic manner.
4	Representing constraints	Implicit constraints about boundary are made explicit.
	~~Label-Retrieval~~ (Replaced: see 5–8 below)	~~Recalling previously solved solutions of partial solutions~~
5	Retrieving domain knowledge	Prior knowledge about problem domain is retrieved.
6	Retrieving technique knowledge	Prior knowledge about modeling technique/method is retrieved.
7	Making meta-level notes	Making notes about managing the process or other meta-aspects.
8	Retrieving from experience base	Drawing on experience the designer is building during this design.
9	Refocusing *(Added)*	Refocusing on different parts of the current design.
10	Validating *(Added)*	Validation to check that the needs are met and solution is consistent.
11	Note making	A note is made of issues that need to be addressed later.

table, was used for the few protocols that were unintelligible or unrelated to the development process (e.g. comments to the administrators of the experiment). The identification and use of additional categories follows the mode of research suggested by (Pare', 1995, p. 554).

The extended model, thus, became the basis further analysis. The 'context,' indicated by the phrases preceding and following the phrase in question, often dictated the coding. In case of difficulties, the coders appealed to first principles, that is, reverted to the source model and/or attempted to find analogies to the phrase in question—in the phrases suggested by the authors (Adelson & Soloway, 1985). An example of the coding results appears in Table 5A.

Each protocol fragment was also coded, during a second pass, to identify use of different object-oriented modeling techniques. The task required use of three object-oriented modeling techniques: use cases, class diagrams and sequence diagrams (Booch, Jacobson, & Rumbaugh, 2001). The protocol fragments were, therefore, mapped to indicate the use of technique. In most cases, the developers provided clear indications about the modeling technique being used. Where such indications

Table 5A
An example of coding results

	Concept Formation	Concept Expansion	Concept Simulation	Representing Constraints	Retrieving Domain K	Retrieving Technique Knowledge	Making Meta	Retrieving from Experience Base	Refocusing	Validating	Note Making	See Table 5: Extended Model of Developer Behaviors - for an explanation of categories
Examples (Andrea)	P=problem space, D=design space											**Explanatory Comments**
After the user has pressed the summon button			*P*									Simulating what the user will do
the system is gonna, the system is gonna light the, light the bulbs			*D*									and the how the system will respond.
So, do I need a class, do I need a class 'summon button bulbs' or 'summon button lights' to indicate that the buttons are on and off?	*D*											Forming a class.
Or it can be an attribute in a summon buttons, summon buttons, an attribute of a summon button that is either on or off.		*D*										Has previously formed the class. Now considering adding an attribute.
Ok, I'll think about it later on.											*D*	Notes to return to this decision later.
Examples (Stanley)	P=problem space, D=design space											**Explanatory Comments**
And when a user presses the summon button, gonna press the up button or the down button			*P*									Simulating the user pressing the up or down elevator button.
And this is gonna create a signal and its gonna go to the scheduler and that contains information about which button is pressed and what floor is that.			*D*									Simulating the system's response to the user pressing an up or down button.
So it's summon button and it would have a signal.	*D*											Decides he needs 'Summon Button'
It would generate a signal, summon.			*D*									Simulates 'Summon Button' generating a signal.

were not available, the context, i.e. surrounding phrases, provided clues to the identification of the technique the developer was using. Table 6 shows the template used for this mapping, and examples.

3.3. Analysis

Following the coding, the analysis proceeded as a hermeneutic process, marked by two broad stages. First, a number of descriptive analyses were performed. This stage yielded descriptions of surface features of the development process (Bush & Purao, 2000). Several displays of the data were created such as frequency count, mapping of behaviors to use of UML techniques, surface maps (Bush & Purao, 2000) and others (Tufte, 1983; Goldschmidt, 1997). These different representations allowed the researchers to understand the data in different ways. More importantly, they allowed frequent checks on different interpretations the researchers attempted based on the data. The use of displays, therefore, allowed performance of careful comparisons, detection and confirming of inferences, noting of patterns and themes and seeing trends (Miles and Huberman, 1984, p. 92).

The second stage involved use of different theoretical bases, prior design studies, and writings on O-O development to arrive at interpretations of behaviors that may be of interest. This, highly iterative, phase involved a sequential strategy, wherein a within-case analysis strategy was followed by an across-case strategy. One of the participants was used to reach initial interpretations about the use and explorations of problem and design spaces. The other participant, then, provided either confirmation or refutation of interpretations suggested by analyzing data provided by the first participant. The data analysis was, in this manner, expected to provide similar, not contrasting results. This literal replication strategy evident in the selection of

Table 6
Coding protocols against modeling techniques

UML Technique		Example
Use Case	Use Case	*Use cases. Elevator summoned, makes a request, elevator arrives at a scheduled floor*
	Actor	*The door is gonna open and the person would come out.*
Class Diagram	Class	*There is a, the elevator class has an array of elevators. …The floor sensor class.*
	Attribute	*Scheduler has to have a pending. Pending array of elevators,*
	Method	*And illuminates and cancels the illumination. Scheduler.*
	Association	*so the relationship between, there is an association between floor and summon button.*
Sequence Diagram	Class	*Do we need this in a class? I know we'll need several sensors. A floor sensor. A weight sensor*
	Existing Attribute/Method	*Its gonna make illuminate.*
	New Attribute/Method	*Then we have press button. Then we have unpress button for the emergency button. Ok, these are classes and methods that I need.*
	Interaction	*I expect the summon button, its gonna call elevator,*
	External Event	*it happens when a user, for example when the user, when summon button is pressed by the user*
	External Response	*Its gonna illuminate after the elevator is summoned, its gonna illuminate the summon button*

participants, who came from a pool of potential or current information systems developers was, thus, followed during the analysis process (see Fig. 1).

The process of shaping propositions is more judgmental in theory-building research because researchers cannot apply statistical tests. The research team is, therefore, required to judge the strength and consistency of relationships articulated. In addition to the literal replication strategy, which can considerably enhance the strength of the relationships articulated, the researchers display the evidence and procedures along with the findings, so the readers may apply their own standards. Consequently, qualitative data is particularly useful in understanding why certain relationships may hold. Having specific examples from the protocol fragments is, therefore, important to allow the reader to judge the findings (Guba & Lincoln, 1989). In addition, appropriate literature on the phenomenon of interest is used to support or refute the interpretations. In our case, the research streams that directly bear on the phenomenon are information systems development, object-oriented systems and design processes. It was necessary, therefore, to find supporting or contrasting literature that would validate or challenge findings that emerged from the data. The next section develops and validates these findings. Each set of findings is presented as emergent from one case, supported or refuted by the other case, and validated or challenged by the technical literature.

4. Findings

The findings are reported in four categories: existence of problem and design spaces, engagement in the two spaces by the developers, their behaviors in the two spaces, and their use of the modeling techniques to structure the two spaces. The findings reported in each section are organized in the following manner. First, relevant features are identified from **case 1**, followed by a search for confirmatory evidence in **case 2**. To further support these findings, appropriate **literature** is cited or mismatches between the findings and the literature are discussed. The writing uses these phrases in bold in each section once to suggest the moves from case 1 to case 2 and then to enfolding literature.

4.1. Existence

Our observations suggest that the developers move within and between a problem space and a design space. They interpret, elaborate upon and attempt to understand

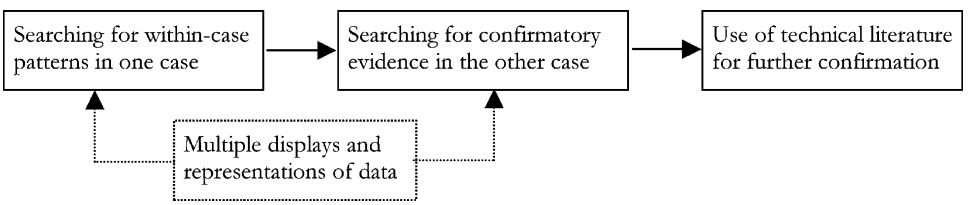


Fig. 1. The analysis process.

the problem; and devise, expand upon and continually adjust a design. Consider the protocol fragments below from **case 1**. Andrea (protocol fragment 5–11): "*It needs to be able to get to the correct floor and has a certain weight limit, has sensors to watch the weight limit. The sensors know when to stop the elevator when it is approaching the floors. It has to illuminate when people know which floor they're going to. Certain floors have to illuminate, up and down has to illuminate when they're pushed.*" and (protocol fragment 454–457): "*And it does that process weight limit and it has been exceeded it does not allow the doors, it does not allow another floor or pending floor pickup to come about until that weight limit has been reduced. It does a, the elevator system does a process move to floor, it does a process direction request. The process move to floor is coming from the destination panel, the destination button. Destination button accepts the floor number.*"

In fragments (5–11), Andrea demonstrates activities within the problem space such as exploring different concepts, considering action sequences, and visualizing a mental representation of concepts. She identifies several required concepts and their properties, such as floor, weight limit and sensor. On the other hand, in fragments (454–7), she demonstrates structuring of the design space, specifically, including decisions about how to handle the weight limit, and detailed decisions such as identification of procedures that must be performed (e.g. process move to floor). She also articulates possibilities for usage of one class by another and considers action sequences among concepts that she has formulated as part of a design to visualize how the artifact, created based on her design, may behave. Activities in the problem space, thus, clarify the environment in which the artifact to be designed must operate and crystallize concepts that must be represented as a part of the design. Activities in the design space, on the other hand, reflect design behaviors such as proposing new concepts, amplifying or adjusting these concepts and refining the design.

Ample examples of similar behaviors were available in **case 2**. Consider protocol fragment (101–2) from Stanley: "*So when elevator arrives at the scheduled floor, its, the floor switch is gonna close and the elevator is, the elevator is gonna stop. After elevator is stopped, the door is gonna open and the person would come out,*" and protocol fragment (168–172): "*scheduler then, that signal would contain of course information about the particular elevator and then a scheduler is gonna stop the elevator at a particular floor. It's gonna send a message to the, it's gonna start a message to the elevator. It's gonna stop and elevator is gonna send a message to the motor to stop and motor is, the floor switch is gonna close and whose gonna send a message to the floor switch?"*

In fragment (101–2), Stanley demonstrates activities in the problem space. He explores the problem space by trying to visualize different actors and their interactions with the system. In fragment (168–172), Stanley demonstrates activities within the design space, such as considering action sequences among concepts that he has formulated as part of a design, and visualizing how the artifact created based on his design may behave. The distinction between problem versus design spaces was, in fact, somewhat difficult to draw as we analyzed and coded the behavior protocols. We attribute this to reasons discussed in section 2.2. leading to the blurring of boundaries between problem and design space. During the protocol analysis this often

became apparent as we could decide on the classification of a protocol fragment only after reviewing the context provided by the surrounding protocols.

The blending of real-world things and concepts to represent these is a recurring theme in the **literature** dealing with object-oriented system development and program design. Kant and Newell (1984) describe this as the 'insight' mechanism, which facilitates a clear mapping between domain and implementation objects. Such mapping and its consequence—the distinction between concepts in the problem and design space—are also made explicit by Mathiassen et al. (2000), who identify real-world things and their abstractions. Mathiassen's ideas map well to the notions from semiotics (Stamper, 1996) of referent and sign as explicated by Sykes and Gupta (2001). The referent refers to real-world things, whereas the sign stands-in for these real-world things. The referent, therefore, lies in the problem space, whereas the sign is created in the design space. Sykes and Gupta suggest an intervening notion between these two, in the form of a 'concept.' We argue that for a developer, the 'concept' represents a bridge between the problem space and the design space. Distinction between the two spaces is, therefore, a consequence of how closely the 'referent' maps to the 'concept,' and how closely the 'concept' maps to the 'sign.' Following the dictates of object-oriented paradigm (Booch, 1994; Booch et al., 2001; D'Souza & Wills, 1999), referents are represented as signs in object-oriented systems using the users' language. Mapping between the two is, therefore, fairly close—leading to a blurring of boundaries between the two spaces. Further support for these arguments is provided by McPhee (1996), who suggests that objects are often anthropomorphized during the development process. Schön (1983), whose ideas underlie McPhee's analysis, also argues for a reflective mode of designing that is facilitated by close mapping between the two spaces during object-oriented development. Developers can, in this manner, "listen" to the design by imagining themselves to be the objects (Schön, 1983).

The blurring of boundaries between the two spaces that the above literature alludes to is exploited by (Skagstein, 2000), who argues for an initial step during the development process to separate 'objects' and their 'alter-egos'. Following his argument, an object or class identified in the models does not 'represent' a real-world object, but rather, designates an augmentation of the corresponding real-world object. In our protocols, we found specific examples of these such as the concept 'Floor Switch,' which straddles the boundary between the problem space and the design space. The referent and the sign, thus, overlap. As Skagstein (2000) argues, the developer, during the development process, adds behaviors and properties to the real-world concept of 'Floor Switch,' that is, augmenting the real-world 'Floor Switch.' Based on the decisions about these properties/behaviors, the developer decides where the boundary would lie between the artifact she is modelling (design space) and the real-world concepts represented in it (problem space). Deciding on the appropriate boundary will, then, lead to the appropriate sharing of responsibilities between the real-world 'things' and those modeled within the system (Skagstein, 2000). The overlapping spaces, therefore, contribute to the fuzziness of the system boundary. Finally, our analysis can also be seen as evidence for Simon's (1974) early intuition about the existence of multiple problem spaces, which in fact are realized in a 'constructive'

manner, as we have shown, within a design space. Table 7 below summarizes these findings.

4.2. Engagement

In spite of the difficulties in deciding between problem and design space, the emphasis on the latter is evident as separation between the two spaces is borne out across multiple sessions. We observe a trend for progressively greater emphasis away from the problem space towards the design space—across the sessions. As the development proceeds, the designers' emphasis quickly shifts from the former to the latter (see Fig. 2) within each session. Averaged across sessions, Andrea shows significant emphasis on design space (87.64%). The largest percentage of problem space activities occurs early in session 1 (26.88%). In the final session, she shows an almost exclusive bias towards the design space with problem space activities accounting for an insignificant fraction (2.11%). Fig. 2, represents **case 1**, that is, Andrea's sessions.

These observations are mirrored in **case 2**, that is, by Stanley's relative focus on the two spaces. Stanley, too, quickly shifts from the problem to the design space across the sessions. Though he spends a slightly higher fraction of time in the problem space in session 1 (32.94% as opposed to Andrea's 26.88%), his emphasis on the design space increases rapidly. His average emphasis in the design space, at

Table 7
Findings Set 1—Existence of problem and design spaces

Findings	Case 1	Case 2	Literature
The distinction between and blurring of problem and design spaces is manifested during ISD with O-O.	Supported	Supported	Kant and Newell, 1984; Mathiassen et al., 2000
Cause: Object-orientation leads to blending of referent and the concept into one sign.	Supported	Supported	Schön, 1983; Sykes and Gupta, 2001
Consequence: Overlapping of spaces leads to blurring of boundaries between the two spaces.	Supported	Supported	McPhee, 1996; Skagstein, 2000

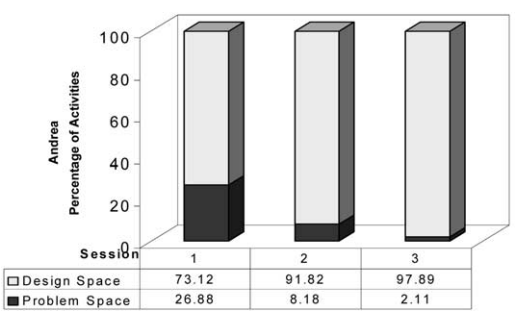


Fig. 2. Case 1—Andrea's relative focus in problem and design spaces.

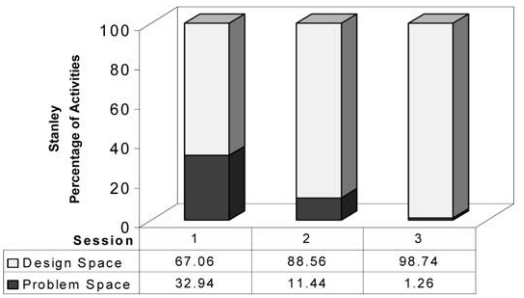


Fig. 3. Case 2—Stanley's relative focus in problem and design spaces.

84.79%, is comparable to Andrea's. Also, like Andrea, he shows an almost exclusive bias towards the design space in the final session. Fig. 3 shows Stanley's sessions.

To further understand whether the designers engaged in each space early or late during the sessions, we recorded the first and last engagement in each space. Fig. 4 shows this analysis for **case 1**.

The figure shows that Andrea starts sessions 1 and 2 in the problem space and ends in the design space. In the figure, the arrows show protocol-fragment numbers for the beginning and end of each session, and the first and last engagement in each space. We may, therefore, characterize Andrea's first two sessions as a start-off in the problem space (protocols 1–12 in session 1), engagement with the design space along with the problem space (protocols 13–21 in session 1), and a move to the design space (protocols 22–76 in session 1). The difference between the second and the first session is that Andrea moves a little faster into the design space. Note that unlike the first two sessions, she *starts* and ends the third session in the design space. If we consider the first move into the design space, Andrea's moves into the design space, then, occur more quickly within each session (after 12 protocol fragments in session 1, after 9 in session 2, and immediately in session 3). Fig. 5 shows the corresponding analysis for **case 2**.

Stanley shows a similar pattern. His moves into the design space also occur faster

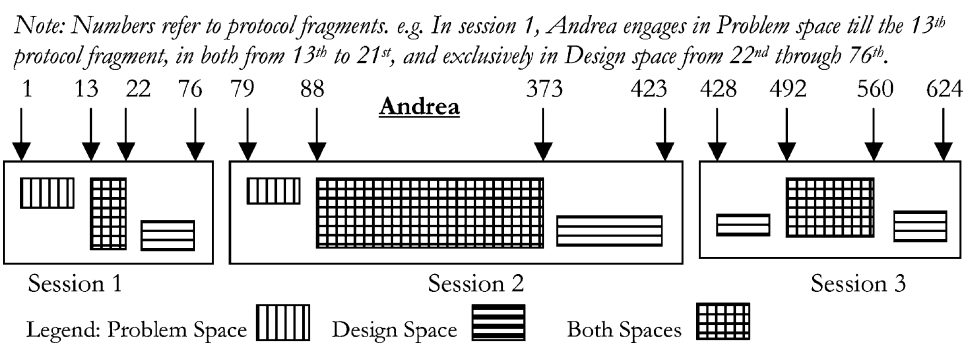


Fig. 4. Case 1—Andrea's engagement with problem and design spaces.

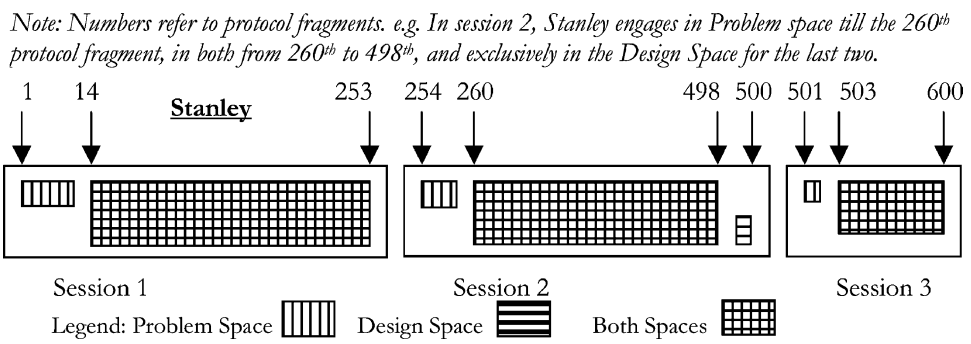


Fig. 5. Case 2—Stanley's engagement with problem and design spaces.

with each session (13, 6, and 2 fragments respectively). Unlike Andrea, who starts the final session in the design space, Stanley starts the final session, albeit briefly, in the problem space. Another difference in the strategies of the two designers is evident from a comparison of Figs. 4 and 5. Andrea's behavior is characterized by exclusive attention to the design space at the end of each session, as opposed to Stanley, who continues to engage with both spaces. Andrea, therefore, appears to follow a breadth-first strategy, covering the problem space before moving on to the design space. Stanley, on the other hand, follows a depth-first strategy addressing parts of the problem space, moving to the design space and then reverting to other parts of the problem space.

Several of our observations are confirmed by prior **literature**. First, it is known in ISD lore that given a deadline, a design emerges, instead of a continuous search for the perfect design. (Verhoef, 1993) observed similar moves with experienced subjects using ORM. It supports the common rule of thumb that 'analysis paralysis' is best avoided by defining clear timeline and deadlines for subtasks. The shift from problem to design space corresponds to methodological prescriptions such as those from the Rational Unified Process (RUP) (Kruchten, 1998), which argue for higher emphasis in design space during later phases. The rapid moves from the former to the latter are, however, contrary to the arguments for real micro-cycles. RUP (Kruchten, 1998) suggests moving between the two spaces in a strict sequential 'mini-waterfall' approach. While Stanley appears close to achieving this, seemingly, with his depth-first approach, Andrea's engagement in the design space becomes increasingly exclusive across the sessions with her breadth-first approach. A possible reason for the rapid increase in emphasis in the design space may be the increased burden that the diagramming techniques, such as class diagrams, pose for the developers (Rossi & Brinkkemper, 1996). As prescriptive writings about O-O development have evolved from O-O programming, the emphasis is still on design and implementation. Engagement in the design space, as opposed to that in the problem space is, therefore, naturally supported in prescriptive models of object-oriented development. A less flattering interpretation of the increasing emphasis in the design space suggests that the developers, having selected a plan of attack, do not return to re-examine their assumptions and instead, focus their efforts on making the initial solutions more

Table 8
Findings Set 2—Developer engagement in problem and design spaces

Findings	Case 1	Case 2	Literature
Emphasis in design space increases rapidly across sessions.	Supported	Supported	Verhoef, 1993; Kruchten, 1998
Cause: Faced with a deadline, developers quickly engage in the design space.	Supported	Supported	Verhoef, 1993
Cause: Availability of diagramming techniques in the design space provides a higher cognitive burden.	Supported	Supported	Rossi and Brinkkemper, 1996
Consequence: Micro-cycles appear as back-and-forth engagements in the two spaces.	Partially supported	Partially supported	Kruchten, 1998
Consequence: Fewer returns to the problem space lead to design fixation.	Supported	Supported	Purcell et al., 1994

complete. This behavior pattern, design fixation (Purcell, Gero, Edwards & Matka, 1994), can lead to inferior designs or designs that cannot be justified since the design process has not considered and evaluated alternative designs. The rather quick change in emphasis into design space observed between sessions 1 and 2—for both developers—indicates that design fixation may, in fact, be occurring. Table 8 below summarizes key findings from the above discussion.

4.3. Behaviors

The developers' exploration of the two spaces can be further characterized using the extended model of behaviors (Table 5). Fig. 6 shows these behavior patterns for **case 1**, Andrea. The horizontal axis indicates the behaviors, and the vertical axis, the frequency with which each behavior was observed. The figure shows the behaviors in each space, for a session, as a separate series of connected points.

From Fig. 6, a few spikes are clearly evident. The most prominent is Concept Expansion (second data point) in the design space, particularly in the second session. This is complemented by a rise (over the first session) in Concept Simulation (third data point) in the design space. The primary focus in this pairing appears to be Concept Expansion, with Simulation as a vehicle. Typically, Expansion leads to

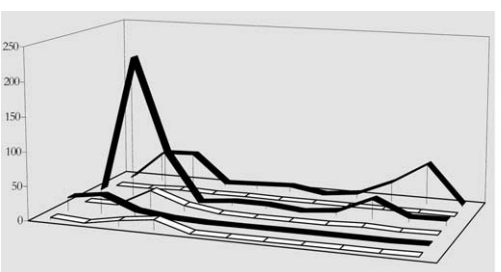


Fig. 6. Case 1—Andrea's behaviors in problem and design spaces.

Simulation, which in turn, leads to further Expansion. Consider the following protocol fragment (156–9): *"Got to have some kind of pending array of elevators, one to forty. And it will place in pending a request, and it will process the request so the elevator can send the correct elevator there. Elevator will have a direction and it is boolean."* Here, Andrea begins expanding her model by asserting that an array of pending elevators may be necessary. From here, she moves to simulate this decision, indicating that it will need to communicate with the Elevator. She then realizes that the Elevator should, therefore, have a direction and decides that it will be a boolean attribute. Clearly, this interplay between expansion and simulation is more relevant for the design space than for the problem space. Another use of Simulation is observed in the design space in the final session. In Fig. 6, the series in the far back (session 3, design space) shows the occurrence of Simulation (third data point). Expansion does not, however, dominate Simulation as it does in the first two sections. Instead, Validation (the data point second from right) dominates Simulation. Consider the protocol fragment (613–6): *"How about signal. It is implicit that the summons button accepts is and it does, it sends back illuminate summons button. Ok, ok going to the elevator system, we send a message to process the direction, process direction with it."* Andrea's primary purpose during this protocol is validation ('*Ok, ok going to the elevator system*'). She uses simulation to help during this process ('*we send a message to process the direction*'). While she simulates to validate, she expands the design further to accommodate any deficiencies revealed by the simulation. For example, consider protocol fragment (573–6): *"And sends a message elevator, elevator sends a message to the summons button. No. No, no, no. Elevator system sends a message to the summons button to, the summons button's got to have a modify status."* Here, Andrea's simulation of the design space results in partial validation and the deficiency she notices (a missing attribute in the 'Summons Button' class) is fixed as she expands concepts in the design space.

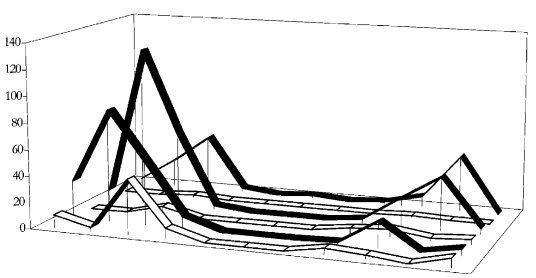
Another significant activity observed in session 1 is concept formation. As expected, when the development process begins, Andrea engages in concept formation in the design space. Consider the protocol fragment (36–9) *"Do we need this in a class? I know we'll need several sensors. A floor sensor. A weight sensor,"* where Andrea identifies several classes. The only other significant spike in the design space is observed for the activity Refocusing (the data point third from right) in the second session. This makes sense because the task is non-trivial and she needs to focus on different parts of the design space. For example, consider the protocol fragment (312–21): *"The summons button has illuminate, its boolean.* (313–317 skipped) *And illuminates and cancels the illumination. Scheduler. Scheduler has to have a pending. Pending array of elevators, one to four."* Here, Andrea demonstrates that having tackled one set of design decisions, about the 'summons button,' she turns her attention to another portion of the space, dealing with the 'scheduler.' Interestingly, there is no corresponding increase in the activity Note Making. This lack of note making when moving from one portion of the design space to another may be attributable to the novice level of the participants.

Andrea's emphases on behaviors in the problem space are quite different from those in the design space. The most significant behaviors in the problem space occur

in sessions 1 and 2. Specifically, she engages in Simulation (third data point in Fig. 6) in an effort to better understand the domain. This activity particularly dominates her use of the problem space during session 2. Consider protocol fragments (253–255): *"Ok. Now, I didn't think about that door open. Keeping those doors open. A red emergency switch stops and holds the elevator."* Here Andrea tries to understand the problem by playing with different entities in the system. During the first session, however, Andrea's exploration of the problem space was dominated by the activity Constraint Representation (fourth data point). Her engagement in the problem space during the first session was, therefore, mostly about what must be addressed by her development efforts. Consider protocol fragment (19–22 in session 1): *"So it has to be concerned about safety. … It has to be concerned with panel inside the elevator, the up and down buttons outside the elevator and the sensors and it must know where elevators at."* Here, Andrea makes decisions about constraints.

Stanley's behavior patterns were analyzed in a similar manner to locate further support or to refute the above observations. Fig. 7 shows this analysis for **case 2**.

For Stanley, the pairing between Expansion and Simulation (the second and third data point) in the design space was evident in sessions 1 and 2. Consider the protocol fragment (29–31): *"But then what should be the attribute of a signal class other than it would contain... So summon buttons gonna send a message to scheduler and scheduler, is it an association between scheduler and summon buttons?"* The interplay between Expansion and Simulation is evident in this fragment. Stanley begins with Expansion (deciding on the attribute of 'Signal' class), moves to Simulation (describing a message from the 'Summons Button' class), and returns to Expansion (deciding on an association between the 'Scheduler' and the 'Summons Button' class). Protocol fragment (106–7): *"I think elevator has motor. I need a class motor,"* is an example of this activity. Like Andrea, Simulation continued to be a significant activity in session 3 for Stanley. However, in that session, expansion did not pair with simulation. Like Andrea, it was validation (data point second from right) that was paired with expansion. Consider the protocol fragment (519–26): *"Choice made. Who would illuminate button? Update choice. So in the destination button and then I need an emergency button. An emergency button is gonna have a status which is boolean which shows whether it is pressed or not. Then we have press button. Then we have unpress button for the emergency button. Ok, these are classes and methods that I need."* Stanley engages in Simulation as he ensures that the appropriate classes



Legend:
Horizontal Axis: Behaviors (see table 5)
Vertical Axis: Frequency of Behaviors
Planar Axis: Sessions 1 to 3 from front to back
Bands: Problem Space (white), Design Space (black)

Example: The highest frequency in the problem space (white band) is for behavior 3 (concept simulation) for session 1 (starting from the front).

Fig. 7. Case 2—Stanley's behaviors in problem and design spaces.

and methods have been captured in his design to fulfill the requirements. As expected, the activity Concept Formation was also a significant activity for Stanley in session 1. Refocusing (the third data point from right) was also a significant activity in session 1 and 2 for Stanley, unlike Andrea, for whom the activity was significant only in session 2. The protocol fragment (67–8): *"So that, I think, gives me something to go on as well as the first use cases. Now, move on to the second use case"* shows an example Stanley's Refocusing.

Stanley's emphases in the problem space were also different compared to those in the design space. Like Andrea, Simulation and Constraint Representation dominated the problem space behaviors. Simulation was particularly noted during the first session as a technique that Stanley used to explore and understand the problem space, as seen in protocol fragment (128–132): *"The view from the floor is like the arrival panel. Have to think about the arrival panel. What's happening there because arrival lights … arrival panel containing one illuminator for each floor number … the number of floor at which elevator is arriving."* The use of Constraint Representation was noted during the first session to clarify what must be addressed by the development task, as seen in the following protocol fragment (110–12): *"The elevator mechanism will not obey any inappropriate, as if the floor switch is not closed when the system issues a stop signal the elevator … So the elevator stops only when the floor switch is checked the floor switch signal and if the floor switch signal is closed then its gonna, the elevator is gonna stop. Otherwise it will disregard the signal."*

The observations above are supported, though partially, by **literature** on prescriptive writings on development processes. Specifically, the pairing of expansion and simulation in the design space is contrary to the rigid prescriptions of RUP, the Rational Unified Process (Booch, Jacobson, & Rumbaugh, 2001) as described by Kruchten, who suggests that each phase in the development process uses a mini-waterfall approach (Kruchten, 1998). Hesse challenges such descriptions as well arguing that rigid phases are not found in modern projects, lending support to our observations (Hesse, 2001). Following the description of RUP and other idealized development processes (e.g. Jacobson et al., 1999), an argument can be made that the pairings we observe are in fact opportunistic adaptations of these prescriptions. On the other hand, these observations may mean that strict methodological dictates (e.g. Jacobson et al., 1999; Kruchten, 1998; Fowler, 1999) cannot be sustained in the face of these emergent patterns. An example is the different use of pairing of behaviors by Andrea and Stanley. Andrea's pairing is mediated by Refocusing in the second session to support her depth-first strategy; Stanley's use of Refocusing was generous in both the first and second sessions to support his breadth-first strategy to development (Bush & Purao, 2000). The other pairing of behaviors—between simulation and validation—can also be interpreted using the abovementioned perspectives. A related perspective is available in Guindon et al. (1986), who identifies breakdowns that occur during the design process. Following her arguments, the pairings between expansion and simulation, and that between simulation and validation may be interpreted as the strategies adopted by the developers to minimize the breakdowns. On the other hand, Nguyen and Swatman (2001) argue that breakdowns are important in problem space exploration. They noticed that the activities of break-

downs force a reconceptualization of the problem space. Schön (1983) comments that developers often start with a rough plan. During the development process, it is tested, and if found faulty, or if limitations arise, the requirements are redefined or the reality under investigation is re-interpreted. He argues that any decisions entail side effects, which can radically affect the viability of proposed solutions. On the other hand new problems encountered during development result in a deeper understanding of the problem space (Schön & Bennett, 1996). The above may also explain why developer behaviors in the problem space differ considerably from those in the design space. As developers explore the problem space or return to the problem space after engaging in the design space, they need to ensure that the framing imposed by their decisions in the design space fits their interpretation of the problem space (Schön and Bennett, 1996). Behaviors such as constraint representation and simulation, observed in both cases in the problem space, lend further support to Schön's arguments. Table 9 summarizes key findings from the above discussion.

4.4. Structuring

The protocols were examined to understand how the developers used the modeling techniques to frame and structure the two spaces. The following examples are characteristic of the kinds of usage demonstrated by Andrea, that is, **case 1**. As expected, use cases provided the most support for exploration of the problem space. Consider, for example protocol fragment (25–32): *"Use cases. Elevator summoned, makes a request, elevator arrives at a scheduled floor, elevator arrives at a nonscheduled floor, elevator ready, elevator overweight, elevator."* The use case 'elevator summoned' in this case, allows Andrea to simulate actors and their interactions, that is,

Table 9
Findings Set 3—Developer behaviors in problem and design spaces

Findings	**Case 1**	**Case 2**	**Literature**
Expansion and simulation occur as paired behaviors in the design space.	Supported	Supported	Contrary to Kruchten, 1998
Simulation and validation occur as paired behaviors in the design space.	Supported	Supported	Contrary to Kruchten, 1998
Cause: The pairings represent opportunistic adaptations of methodological prescriptions.	Supported	Supported	Hesse, 2001
Cause: The pairings represent strategies adopted by the developers to minimize breakdowns.	Supported	Supported	Nguyen and Swatman, 2001; Schön, 1984
Behaviors in the problem space differ considerably, primarily focusing on constraint representation and simulation.	Supported	Supported	Nguyen and Swatman, 2001; Schön and Bennett, 1996
Cause: Problem space behaviors represent (re-)interpreting the problem space based on decisions in the design space.	Supported	Supported	Schön and Bennett, 1996

the problem space. Andrea also uses the same technique for a gradual move to the design space. As seen in the above fragment, simulation in the problem space gives way to identification of different states the elevator may be in, which must be recognized by the system being developed. As opposed to the use cases, which primarily support exploration of the problem space, the class diagram allows shaping the decisions in the design space. Consider, for example, protocol fragment (49–60): *There is a, the elevator class has an array of elevators. …The floor sensor class. The destination button class. The summon button class. The summon light class. The destination button will be an aggregate button in the elevator class. …The floor sensor is part of the elevator so its an aggregate."* Andrea identifies a number of classes, which she goes on to structure in relation to one another. These decisions provide a frame of reference that Andrea uses to confirm or further interpret the problem space on later occasions. Consider, for example, fragment (245–46): *"The floor sensor knows it's the floor and makes the elevator stop. So the motor uses the floor sensor to stop."* Here, Andrea uses the concept of the floor sensor class she has devised earlier to predict how the eventual implementation of the system (which includes her decisions about representing the problem space) will interact with events in the problem space. Finally, the interaction diagrams support a variety of activities in both spaces. There is considerable mingling of the two spaces when the interaction diagram is used. Consider, for example, protocol fragment (451–4): *"And it does that process weight limit and it has been exceeded it does not allow the doors, it does not allow another floor or pending floor pickup to come about until that weight limit has been reduced. It does a, the elevator system does a process move to floor, it does a process direction request. The process move to floor is coming from the destination panel, the destination button. Destination button accepts the floor number."* The protocol clearly begins in the design space as Andrea alludes to 'it' doing a process, referring to the system being developed. While she describes the intended interaction as she sees it in the design space, she freely returns to the problem space, sometimes inadvertently, at other times to ensure that the interaction between the problem and design spaces is accurately captured.

Similar use of modeling techniques was observed as Stanley (**case 2**) engaged in the two spaces. For Stanley as well, use cases were useful in exploration of the problem space, as evidenced by protocol fragment (101–103): *"So when elevator arrives at the scheduled floor, its, the floor switch is gonna close and the elevator is, the elevator is gonna stop. After elevator is stopped, the door is gonna open and the person would come out. Do I need to have a class for this because its seemingly its changing the transition."* The use case 'elevator arrives at a floor' allows Stanley to simulate the actor's interaction with the system and facilitates a gradual move to the design space. For example, based on the use case, Stanley questions whether a class is needed. The class diagram, as expected, was useful in forming and exploring the design space. For example, in protocol fragment (54–61), Stanley engages in the design space to structure classes: *"Floor has a summon button so the relationship between, there is an association between floor and summon button. Or is it an aggregation? I think its an association. So if the floor, the floor has summon button and an elevator has destination button. And the floor can have a minimum of one summon*

button and a maximum of two and each summon button corresponds to a single floor." The classes provide a frame of reference that Stanley then uses when he returns to the problem space, for example, when he identifies the correspondence between the summon button and floor. This propensity to use design space decisions to interpret the problem space is also seen in the protocol fragment (107–111): *"I need a class motor. It has, I think, three states, stop moving and, stop moving up and moving down. So, so when the elevator stops, its gonna send a signal to the motor to stop it. The elevator mechanism will not obey any inappropriate, as if the floor switch is not closed when the system issues a stop signal the elevator … So the elevator stops only when the floor switch is checked the floor switch signal and if the floor switch signal is closed then its gonna, the elevator is gonna stop."* Here, Stanley uses the classes 'motor' and 'floor switch' that he has identified to understand how the elevator will function when it is controlled by his planned system, which includes these classes. Finally, interaction diagrams are used by Stanley to engage in the two spaces in a variety of ways. For example, consider the fragment (541–545): *"Interaction diagram for the use case elevator summoned. So it happens when a, it happens when a user, it happens when a user, for example when the user, when summon button is pressed by the user, its gonna be, gonna press the summon button so, I expect the summon button, its gonna call elevator, its gonna call elevator, its gonna call elevator summon. Its gonna make illuminate. Its gonna illuminate after the elevator is summoned, its gonna illuminate the summon button."* Stanley slides from the problem to the design space fairly easily, using classes and methods from the system he is developing.

These uses of the modeling techniques in the two spaces map well to uses suggested by the prescriptive **literature** about system development with O-O. First, use cases provide developers the ability to identify chunks of functionality expected from the system. Jacobson, Christerson, Jonsson and Overgaard (1992) describe this as typical use situations for a system. The use cases in their essential forms (Larmon, 1997) are described in terms of the users' language and expectations, that is, drawing on the problem space (Jacobson et al., 1992). Such use of the use cases—to explore the problem space—has also been observed for experienced designers (Dawson & Swatman, 1999). More specifically, specification of the use case includes steps, some of which may be in the problem domain and others in the system being designed (Jacobson et al., 1994). Kruchten (1998) suggests that use cases, in this manner, provide a bridge that unites problem and design space explorations. Our observations are, therefore, strongly supported by several prescriptive writings about O-O methodologies. Class diagrams, on the other hand, provide ways to capture and describe decisions in the design space (Dawson and Swatman, 1999). Class diagrams are suggested as the primary vehicle to represent and structure concepts that will be eventually implemented in the proposed system (Booch, 1994). Initial decisions the developers make about classes and their structuring provide the perspective developers use as they return to explore the problem space (Nguyen & Swatman, 2001). These descriptions are borne out by our observations. Finally, interaction diagrams contain constructs that allow developers to visualize message passing among objects (D'Souza & Wills, 1999). Explorations of the design space (such as simulations)

are, therefore, expected to be supported by interaction diagrams (Fowler, 1999). Our observations are confirmed by these writings. Another suggested use of the interaction diagrams is validation of the functionality realized within the system against the expected functionality (D'Souza & Wills, 1999). Use of interaction diagrams, in this manner, allows returning to the problem space—for checking decisions in the design space against expectations articulated in the problem space (Kruchten, 1998). Our findings are, therefore, supported by the above literature. Table 10 below summarizes key findings from the above discussion.

5. Conclusions

The findings reported in the paper revolve around the importance of distinction between problem and design spaces, and developers' explorations of these spaces during system development with the O-O paradigm. We find that there are sufficient differences between these two spaces to warrant a separate consideration and explicit recognition of the two during IS development processes.

5.1. Contributions

The central contribution of the study is a grounded description of a specific aspect of the ISD process—the existence and use of problem and design spaces by developers. The grounded theory approach we have employed is supported by prescriptive writings and informal observations about object-oriented systems, development processes and modeling techniques. Specifically, we find that developers engage in the two spaces with different techniques and spend varying amounts of time in each. The

Table 10
Findings Set 4—Structuring problem and design spaces

Findings	**Case 1**	**Case 2**	**Literature**
Use cases provide ways to consider chunks of functionality in the problem space that must be realized in the design space.	Supported	Supported	Jacobson, 1992; Dawson and Swatman, 1999
Consequence: Use cases provide a bridge for a gradual move into the design space.	Supported	Supported	Kruchten, 1998
Class diagrams primarily allow structuring of concepts in the design space.	Supported	Supported	Booch, 1994
Consequence: Class diagrams provide a frame that the developers use to (re-)interpret the problem space.	Supported	Supported	Nguyen and Swatman, 2001; Schön and Bennett, 1996
Interaction diagrams primarily allow exploration of the design space using terms suggested by the problem space.	Supported	Supported	Fowler, 1999
Consequence: Interaction diagrams provide ways to validate decisions in the design space.	Supported	Supported	D'Souza et al., 1999

micro-cycles we observe within and across these spaces represent opportunistic deviations from the prescribed methodologies. In spite of such moves, we observe design fixation, which has been known to be a problem in ISD lore for some time but not accounted for by prescriptive micro-processes embedded in models or tools. The developers' use of different modeling techniques to frame and structure the spaces largely matches the prescriptions. Developer behaviors in the two spaces, on the other hand, seem to be far more opportunistic as they move between the two spaces several times during each session.

Most descriptions of ISD processes to date have dealt with small tasks, allowing greater control to the researcher during analysis. In this study we collected protocols spread over three sessions. We suspect that the larger task and the consequent data collection over multiple sessions has led to the surfacing of our findings. Since some of our observations do find support in prior work (e.g. Guindon, 1990), our research clearly represents an extension of the research stream on design studies. As larger software packages cannot be designed in single sessions, our observations are relevant to understand developer activities across multiple sessions. If the 'fit' between the problem domain and the solution is seen as a crucial quality requirement (Mathiassen et al., 2000), it is obvious that an explicit understanding of the separation between problem and design spaces should lead to better software specification methods and therefore, better systems.

Further, the analysis reported in this paper represents a first effort to identify and analyze these behavior patterns in the context of practically applied software development methods, such as UML. We claim this to be an important contribution as most prior research has used limited domains, such as visual languages, or program code slices. An ancillary contribution to studies on design research is the extension to the model of developer behaviors, suggested by Adelson and Soloway (1985). The additional behaviors (important for ISD using formal, or semi-formal, specification techniques) deal with retrieving prior knowledge about the domain, prior knowledge about the method refocusing, and validating—among others. Adelson and Soloway (1985) and Guindon (1990) allude to these but do not account for them explicitly. Another new behavior, making meta-level notes, may be important for experienced designers (Rossi & Tolvanen, 2000).

The selection of small number of developers does remain a limitation of our findings. However, as design activities are messy, the insights provided by studies similar to ours can be more meaningful instead of obtaining quantitative evidence in controlled experiments if the task at hand is of realistic complexity. The credibility of our findings is supported by two sources. First, use and extension of a behavior classification scheme suggested earlier (Adelson & Soloway, 1988) has provided our analysis with a rigorous foundation. Second, the observations we have reported in the paper were corroborated by multiple analyses and supported by expected or prescriptive suggestions from other researchers.

5.2. Implications

The observations we have discussed, and their interpretations in the context provided by existing literature, have interesting implications for ISD methods, process models, tools and education practices. First, we notice the importance of 'system boundary,' affected by the two spaces. This is especially relevant for embedded systems, where the objects of interest represented in the design and those in the problem space may have significant overlap. Drawing on what we observe as difficulties in distinguishing the problem and design space, we suggest that development methods be enhanced to give more precise guidelines for using different models, especially during early stages of ISD. We note that the propensity to move between spaces without recognizing it can be dangerous, especially for less experienced developers, leading to inadvertent sliding from analysis of the problem into details of the design. We recommend enhancements to educational practices to stress this for the novices.

Second, few behaviors in the problem space are supported by the O-O modeling techniques. While developers did use the use cases, several behaviors in the problem space involved reading the requirements and running through scenarios informally. The important behavior of Simulation in the problem space (van Zutphen & Mantelers, 1996) was not well supported by any of the techniques. While the techniques do support requirements capture, this support does not include notions such as expansion or simulation of concepts in the problem space. This lack of support for problem space exploration has been recognized by other researchers. Rosson (1999) argues that there is a need to add support for problem space exploration in the form of stakeholder scenarios (Rosson, 1999). Further, in their study of CASE tool support for UML, Damm et al. (2000) notice the lack of techniques and notations for initial phases of software development.

Third, we call for additional support to facilitate developers' movements across these spaces. The current spate of tools does not explicitly support movement between problem and design spaces. It may be possible, for instance, to use techniques such as hypertext and cross-referencing for moving between spaces and models. Cronholm and Goldkuhl (1994) comment that it should be easy to generate several alternative solutions within a reasonable time frame. These could be potentially very useful for exploring the boundaries of the system and the consequences that different solutions have for this boundary. The rapid generation of multiple alternatives may also alleviate problems such as design fixation.

Fourth, few process-support mechanisms or tools are currently available to guide the designer through micro-level tasks. On the other hand, as the opportunistic and ad hoc movement between spaces is frequent, a tool, which would enforce a strict process, could be seen as an obstacle in the development process. Such tools or support mechanisms should, therefore, assist the developers in planning tasks with techniques such as agenda management as well as for iterating core processes by enforcing a minimal set of prescriptive tasks.

Fifth, the paired use of expansion and simulation shows that simulation is an important behavior that developers use to expand and develop the design. To

accommodate this, the current, mostly static, tools for recording designs (CASE) could be improved to allow for simulation especially during the early conceptual stage, instead of waiting for the detailed specifications to emerge. We believe that this should lead to tools, which offer support for non-structured early requirements gathering and smooth transitions from non-structured to structured models (Damm et al., 2000). Such use of simulation of the problem space is often observed, and other researchers have suggested designing tools to support this activity (van Zutphen & Mantelers, 1996). A related issue is support for the validation activity with simulation. One implication of this may be to improve the available tools to incorporate design rationale capture as an integral part of the development process (Ramesh, Powers, et al., 1995) or forcing validation as a separate activity during ISD. The tools may also be enhanced to support peer review or allow translation of the design into a textual description that can be validated by mapping it back to the requirements.

As the discussion suggests, Problem and Design spaces are a potent perspective to understand and improve the practice of ISD. The findings reported in this paper along with the implications identified above represent our initial explorations to define and outline these aspects of the ISD process. Our current work focuses on leveraging these findings to create methodological or tool-based support for these aspects of ISD.

Acknowledgements

The paper has benefited from feedback on earlier drafts by Dan Robey, Mark Keil, and Vijay Vaishnavi. We also acknowledge influences of discussions with several colleagues including Rivka Oxman, Lars Mathiassen, Kalle Lyytinen, and Bala Ramesh. We acknowledge the influence of considered and thoughtful comments from the reviewers as well as the associate editor.

References

Adelson, B., & Soloway, E. (1985). The Role of Domain Experience in Software Design. *IEEE Transactions on Software Engineering, SE-11*(11), 1351–1360.

Adelson, B., &Soloway, E., 1988, A Model of Software Design, In M. Chi, R. Glaser, & M. Farr, (Eds.), *The Nature of Expertise*. NJ L. Erlaum press.

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The case study research strategy in studies of management information systems. *MIS Quarterly, 11*(3, September), 369–385.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications, 2nd ed.* Redwood City: The Benjamin/Cummings Publishing Company, Inc.

Booch, G., I. Jacobson, and J. Rumbaugh, "UML Resource Center," http://www.rational.com/, 2001.

Buchanan, R. (1992). Wicked problems in design thinking. *Design Issues, 8*(2), 5–22.

Bush, A., & Purao, S. (2000). Mapping UMLTechniques to Design Activities. In K. Siau, M. Rossi, & P. A. Hershey (Eds.), *Information Modeling in the New Millenium*. Hershey, PA: Idea Group Publishing.

Coad, P., North, D., & Mayfield, M. (1995). *Object Models: Strategies, Patterns, and Applications*. Englewood Cliffs, NJ: Yourdon Press (Prentice-Hall).

Cox, B.J. 1984. Message/Object Programming: An Evolutionary Change in Programming Technology, *IEEE Software*, vol. 1, no. 1, pp. 50-61.

Cronholm, S. and G. Goldkuhl. 1994. Meanings and motives of method customisation in CASE environments—observations and categorizations from an empirical study. *Proceeding of the fifth workshop on the next generation of CASE tools*. B. Theodoulidis. Twente, University of Twente: 67-79.

Cross 1999. *Knowing and Learning to Design*, Conference Proceedings, Atlanta, GA.

Curtis, B., Krasner, H., & Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, *31*(11), 1268–1287.

Damm, C. H., Hansen, K. M., Thomsen, M., & Tyrsted, M. (2000). Creative Object-Oriented Modelling: Support for Intuition, Flexibility, and Collaboration in CASE Tools. In *ECOOP—European Conference on Object-Oriented Programming*. Sophia Antipolis and Cannes: Springer-Verlag.

Dawson, L. L., & Swatman, P. A. (1999). *The use of Object-Oriented Models in Requirements Engineering: A Field Study*. Charlotte, NC: International Conference on Information Systems.

Dorst, K. (1997). *Describing Design: A Comparison of Paradigms*. Technical University of Delft.

D'Souza, D. F., & Wills, A. C. (1999). *Componenets, and Frameworks with UML: The Catalysis Approach*. Reading: Addison Wesley Longman Inc.

Eisenhardt, K. M. (1989). Building Theories from Case Research. *Academy of Management Review*, *14*(4), 532–550.

Ericsson, K. A., & Simon, H. A. (1993). *Protocol Analysis: Verbal Reports as Data (Revised Edition)*. Cambridge: The MIT Press.

Fowler, M. (1999). *UML Distilled: Applying the Standard Object Modeling Language*. Reading, MA: Addison Wesley Longman.

Gane, C., & Sarson, T. (1979). *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall.

Gero, J. S., & McNeill, T. (1997). An Approach to the Analysis of Design Protocols. *Design Studies*, *19*(1), 21–61.

Glaser, B. G. (1992). *Basics of grounded theory analysis*. Sociology Press: Mill Valley, CA.

Goldschmidt, G. (1997). Capturing Indeterminism: Representation in the Design Problem Space. *Design Studies*, *18*(4), 441–445.

Guba, E., & Lincoln, Y. (1989). *Fourth Generation Evaluation*. Newbery Park, CA: Sage Publications.

Guindon, R. D. (1990). esigning the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, *5*(1990), 305–344.

Guindon, R., Krasner, H., & Curtis, B. (1986). Breakdowns and Processes During the Early Activities of Software Design by Professionals, In G. Olson, E. Soloway & S. Sheppard (Eds), *Empirical studies of Programmers Vol. 2*. pp 65–82, 1987. NJ, Abex, Norwood.

Hesse, W. (2001). RUP: A process model for working with UML. In K. Siau, & T. Halpin (Eds.), *Unified Modeling Language: Systems Analysis, Design and Development Issues* (pp. 61–74). Hershey, PA: Idea Publishing Group.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Reading, USA: Addison Wesley Longman.

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach, 4th ed.* Addison-Wesley.

Kant, E., & Newell, A. (1984). Problem Solving Techniques for the Design of Algorithms. *Information Processing and Management*, *28*(1), 97–118.

Kruchten, P. (1998). *The Rational Unified Process*. Reading, MA: Addison-Wesley.

Larmon, C. (1998). *Applying UML and patterns*. NY: Prentice-Hall.

Lyytinen, K. (1987). Different Perspectives on Information Systems: Problems and Solutions. *ACM Computing Surveys*, *19*(1), 5–46.

Mathiassen, L., & Munk-Madsen, A. et al. (2000). *Object Oriented Analysis & Design*. Aalborg, Denmark: Marko Publishers.

McPhee, K. (1996). *Design Theory and Software Design*. Edmonton, CA: The University of Alberta.

Miles, M. B., & Huberman, A. M. (1984). *Qualitative Data Analysis*. Beverly Hills, CA: Sage Publications.

Newell, A., & Simon, H. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice Hall.

Nguyen, L. and P. A. Swatman. 2001. Managing the Requirements Engineering Process. Seventh International Workshop on Requirements Engineering: Foundation for Software Quality, Interlaken, Switzerland, University of Essen.

Oxman, R. (1997). Design by re-representation: a model of visual reasoning in design. *Design Studies*, *18*(4), 329–347.

Pare', G. 1995. *Understanding the Dynamics of IT Implication The Case of Clinical Information Systems*. Doctorial Dissertation. Florida International University.

Pare', G., & Elam, J. (1997). Using Case Study Research to Build Theories of IT Implementation. In A. S. Lee, J. Liebenau, & J. I. DeGross (Eds.), *Information Systems and Qualitative Research*. London: Chapman and Hall.

Pohl, K. (1994). Three Dimensions of Requirements Engineering: framework and its application. *Information Systems*, *19*(3), 243–258.

Purcell, T., Gero, J., Edwards, H., & Matka, E. (1994). Design fixation and intelligent design aids. In J. S. Gero, & F. Sudweeks (Eds.), *Artificial Intelligence in Design*. Kluwer Publications.

Ramesh, B., T. Powers, et al. 1995. Implementing requirements Traceability: A case study. *The Second IEEE International Symposium on Requirements Engineering*, IEEE Computer Society Press.

Rossi, M., & Brinkkemper, S. (1996). Complexity Metrics for Systems Development Methods and Techniques. *Information Systems*, *20*(2), 209–227.

Rossi, M., Tolvanen, J. -P., Ramesh, B., Lyytinen, K., & Kaipala, J. (2000). Method Rationale in Method Engineering. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*. Maui, HI: IEEE Press.

Rossi, M., & Siau, K. (Eds.). (2001). *Information Modeling in the New Millennium*. Hershey, PA: Idea Group Publishing.

Rosson, M. B. (1999). Integrating Development of Task and Object Models. *Communications of the ACM*, *42*(1), 49–56.

Rowe, P. G. (1987). *Design Thinking*. Cambridge: The MIT Press.

Schön, D. (1983). *The Reflectice Practitioner*. New York: Basic Books Inc.

Schön, D., & Bennett, J. (1996). Reflective Conversation with Materials. In T. Winograd (Ed.), *Bringing Design to Software*. New York: ACM Press.

Sen, A. T. (1997). The Role of Opportunism in the Software Design Reuse Process. *IEEE Transactions on Software Engineering*, *23*(7), 418–436.

Simon, H. A. (1979). Problem Solving and Rule Induction. In *Models of Thought*. Yale University Press.

Simon, H. A. (1996). *The Sciences of the Artificial, 3rd ed.* Cambridge: The MIT Press.

Skagstein, G. (2000). Are Use Cases Necessarily the Best Start of an OO Development Process? In *Proceedings of Information Systems Development (ISD 2000) Conference*. Norway: Kristiansand.

Stamper, R. (1996). Signs, Information, Norms and Systems. In B. Holmqvist, & P. B. Andersen (Eds.), *Signs at Work*. Berlin: DeGruyter.

Strauss, A. L., & Corbin, J. (1990). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Newbury Park, CA: Sage Publications.

Sykes, J. A., & Gupta, P. (2001). UML Modeling Support for Early Reuse Decisions in Component-Based Development. In K. Siau, & T. Halpin (Eds.), *Unified Modeling Language: Systems Analysis, Design and Development Issues* (pp. 75–88). Hershey, PA: Idea Publishing Group.

Tan, M. E. (1994). Establishing Mutual Understanding in Systems Design: An Empirical Study. *Journal of Management Information Systems*, *10*(4), 159–182.

Tufte, E. R. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press.

van Zutphen, R., & Mantelers, J.M.M. (1996) Computational Design: Simulation in Virtual Environments, in *Proceedings of the 3rd Design and Decision Support Systems in Architecture and Urban Planning Conference*, Spa, Belgium.

Verhoef, T. F. (1993). *Effective Information Modelling Support*. Technical University of Delft.

Visser, W. (1990). More or Less Following a Plan during Design: Opportunistic Deviations in Spefication. *International Journal of Man-Machine Studies*, *33*, 247–278.

Vitalari, N. P. (1985). Knowledge as a Basis for Expertise in Systems Analysis. An Empirical Study. *MIS Quarterly*, *9*(3),

Wegner, P. (1997). Why Interaction Is More Powerful Than Algorithms. *Communications of the ACM*, *40*(5), 80–91.

Yin, R. K. (1984). *Case Study Research: Design and Methods*. Beverly Hills, CA: Sage Publications.

Yourdon, E. (1995). *Mainstream Objects: An Analysis and Design Approach for Business*. Upper Saddle River: Yourdon Press.

Yourdon, E., & Argila, C. (1996). *Case Studies in Object Oriented Analysis and Design*. Upper Saddle River: Yourdon Press Computing Series.