DOMAIN UNDERSTANDING IN AGILE INFORMATION SYSTEM DEVELOPMENT – THE ROLE OF CONCEPTUAL MODELING

by

Michael Wufka

Diplom-Informatiker, University of Technology, Darmstadt (Germany), 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Business Administration)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

November 2013

Abstract

Agile development methods such as Extreme Programming or Scrum typically do not prescribe specific conceptual modeling practices. In fact, the agile paradigm generally discourages extensive use of documentation and models that are not directly related to or used to generate code. However, anecdotal empirical evidence suggests that conceptual modeling might be beneficial in agile software development, and that a wide range of conceptual modeling methods might be used by agile practitioners.

This work studies the relationship between conceptual modeling and agile development in three parts. Firstly, based on the existing literature, it examines theoretically which role conceptual modeling might play in agile development. In order to do this, it defines an operationalizable conceptualization of agility that is independent of specific development methods. In a second step, the role that conceptual modeling practices play in current information system (IS) development is studied in a survey of practitioners. This survey also serves to validate a measurement instrument based on the conceptualization of agility. Finally, the impact of conceptual modeling on agile development is studied in detail in an exploratory lab experiment. The survey finds that developers who are more agile tend to use a wider range of conceptual modeling practices than less agile developers, and that they face fewer difficulties with domain understanding. The proposed measurement instrument of agility correlates well with self-reported agility, which provides some evidence for its validity and reliability. The results of the experiment indicate that the use of conceptual modeling does not reduce agility, but that it might in fact increase it. It finds that subjects use conceptual models to improve their understanding of the domain in a number of different ways.

These different uses, an interesting result by themselves, are a potential starting point for future research, which could focus on a subset of them. Additional important results of this work include a validated measurement instrument of agility that is based on a development method independent conceptualization of agility, and an improved understanding of the use of conceptual modeling in current agile development practice.

Preface

The research described in this thesis has been conducted by the author. With advice and consultation provided by the supervisor and the committee, the student was fully responsible for identifying and designing the research program, conducting the empirical studies, and analyzing and reporting the results in this thesis.

This work has not been published yet, but it has been presented at various research workshops.

The empirical parts of this work have been approved by the UBC Behavioural Research Ethics Board with certificates H10-02657 and H12-02467.

Table of Contents

List of Tables

List of Figures

Acknowledgements

This research was in part supported by the Natural Sciences and Engineering Research Council of Canada.

I would like to thank my supervisor Yair Wand and my committee members Gail Murphy and Ron Cenfetelli for their generous advice and support.

This work has also greatly benefitted from feedback the author received on presentations of this work on various different occasions, including the research workshop at the MIS division at the Sauder School of Business, the annual SIGSAND symposium, and the ER conference doctoral consortium. I would like to thank the graduate students, faculty and staff in the MIS division for always making their expertise available, and for providing a welcoming and intellectually stimulating environment.

Finally, I will always be grateful for the support I received throughout my academic journey from my parents, Christine and Herbert, my brother Stefan, and my fiancé Karina, who never stopped believing in me, and without whom I could not have completed this work.

For those who ask the interesting questions, rather than the simple ones.

Chapter 1: Introduction

1.1 Research Motivation

Since its emergence around the year 2000 (Beck (1999), Highsmith and Alistair (2001)), the paradigm of agility in the context of information systems development has rapidly gained importance and acceptance. For example, a major market research firm found that in 2009 over one third of all organizations reported that they had adopted practices that fall into the range of agile development (West and Grant 2010).

The emergence of agile development methods can be traced back to several different reasons. Three main influences are listed in (Boehm 2006): The emergence of more rapid change, a counter-movement to previously increasing "process bureaucracy", and human factors. The first reason, increasingly rapid change, was triggered partially by technological progress which led to products with ever shorter lifecycles, e.g., in online and mobile applications. Process bureaucracy describes the previous tendency to strive for more standardization, often in the context of maturity models (Humphrey 1988), as an answer to unsuccessful development projects. Finally, human factors refers to the growing understanding that information systems are complex socio-technical systems (Kling and Lamb 1999), in which emotional and social aspects cannot be neglected.

The agile paradigm in software development is driven by practitioners. One of its earliest and most explicit expressions is in the so-called agile manifesto (Beck, Beedle et al. 2001), which proclaims that its authors have come to value "Individuals and interactions over processes and tools", "Working software over comprehensive documentation", "Customer collaboration over

contract negotiation" and "Responding to change over following a plan". Based on these general principles, a number of different agile development methods have been developed, for example Extreme Programming or XP (Beck 1999) and Scrum (Rising and Janoff 2000).

Independently of the method used, in order to develop information systems successfully, it is necessary that the individuals involved in the development project develop an understanding of the domain of the information system (Khatri, Vessey et al. 2006). Proper domain understanding is believed to be a key factor for successful system development (Offen 2002).

Conceptual modeling is an approach to support domain understanding. Mylopoulos (1992) defines conceptual modeling as "the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication". In other words, any description – textual, graphical or otherwise – of relevant aspects of a domain of interest constitutes a conceptual model. As Wand and Weber (2002) point out, in the context of system development, conceptual models support communication between developers and users, improve understanding of the domain, provide input for system design, and help documenting original requirements. It is important to draw a distinction between conceptual models and design models, which are not in the focus of this work. Design models are another common type of model in systems development, and they are used to describe artefacts constructed during development. For example, a design model might reflect the table structure of a database or the decomposition of source code into modules. Such models are not considered conceptual models, because they do not describe the domain, but the system under construction.

Typically, development methods that are labelled as "agile" do not prescribe conceptual modeling practices. This is not surprising, because the agile paradigm discourages significant

amounts of documentation and other kinds of overhead and instead focuses on source code as the most important project deliverable. To compensate, other approaches to domain understanding are emphasized. For example, Extreme Programming requires the permanent availability of a domain expert (customer), and relies on continuous communication between developers and customer and knowledge sharing between developers, encouraged through programming in frequently changing pairs of developers (Beck 1999).

However, it does not follow that conceptual models are never used in agile development practice. Development methods are typically adapted for each specific project, and many agile development methods explicitly encourage this. For example, Extreme Programming calls for regular team meetings in which potential shortcomings of the development process are discussed and improvements decided (Beck 1999). There is at least anecdotal evidence that such adjustments related to modeling and documentation practices do in fact happen in practice: Hansson, Dittrich et al. (2006) study several projects in detail and find that documentation of for instance requirements specifications are frequently seen as a weakness of typical development practices, and that different projects find different solutions to address their problems in this area.

In this situation, two important questions about the role of conceptual modeling in agile IS development are unanswered:

Firstly, what role does conceptual modeling play in current agile development practice?
Secondly, what role should conceptual modeling play in practice?

Both questions are important for our understanding of IS development practices and for the role of conceptual modeling in domain understanding in the context of agile development. They also carry significant practical importance: Given the limited guidance that agile development

methods provide with regard to conceptual modeling and other practices that might help with domain understanding, practitioners could benefit from simply knowing what other developers using similar development methods are doing in practice. Even more so, any investigation into potential benefits of conceptual modeling in an agile context could inform developers and project managers about what kind of practices to consider to improve domain understanding in their development projects and consequently to improve project success. Due to the large economic importance of IS development, and the increasing portion of development that uses agile development methods, even very modest improvements in project outcomes would be a very worthwhile outcome.

1.2 Agile Information System Development

The literature on agile software development methods is still at an early and immature state. In the aptly named paper "'Lots done, more to do': the current state of agile systems development research", Abrahamsson, Conboy et al. (2009) point to four major gaps in the current literature: A proper understanding of what constitutes agility, the need to extend the applicability of agile methods towards larger and more complex projects, a lack of proper understanding of agile methods beyond their adoption stage, and the scarcity of rigorous empirical studies. The authors of (Dyba and Dingsoyr 2009) agree with this assessment. In a literature review of almost 2,000 studies about agile development (Dyba and Dingsoyr 2008), they assess only 36 to be sufficiently rigorous and relevant to contribute towards answering fundamental questions about the benefits and limitations of agile development, and with resulting implications for research and practice. They also criticize that most empirical studies (39%) employ just a single case study, while only a small number of studies use experiments or a combination of methods (9% and 6%, respectively).

Agerfalk, Fitzgerald et al. (2009) performed a Delphi study on flexible and distributed IS development topics. They conclude that some very fundamental research questions about agile development still need to be answered satisfactorily, including a proper understanding of fundamental concepts such as agility, the appropriateness of agile methods in different situations, the linkage between agile methods and their practices and project success, and questions at an organizational level, such as how organizations select, adopt and benefit from agile methods and agility.

In summary, the literature on agile development is at a very immature state, characterized by very little strong theory, few rigorous empirical studies, and limited cumulative research (Conboy 2009). Due to the young age of the phenomenon of agile development (just over a decade) and the research about it, this state of the field is not surprising. Going forward, work on fundamental concepts of agility and more rigorous empirical studies would be particularly valuable.

1.2.1 Defining Agility

In order to answer the questions posed above in an empirical way, the meaning of the concept of agility in the context of IS development first has to be established. Specifically, how can we determine whether a specific software development project should be classified as "agile" or "not agile"? Only then is it possible to investigate what role conceptual modeling plays in agile development, and what role it should play.

The term "agile" can be defined as "marked by ready ability to move with quick easy grace [...]; having a quick resourceful and adaptable character" (Merriam-Webster's Online Dictionary [1]) or

[1] http://www.merriam-webster.com/dictionary/agile, accessed on Nov. 19, 2013

"able to move quickly and easily" (Compact Oxford English Dictionary [2]). Based on such definitions, agility is a gradual property of software development methods or projects. In other words, a specific method or project is not "agile" or "not agile" in a binary way, but more or less agile, depending to what degree it allows (in the case of a method) or is able (in the case of a project) to move quickly, adapt to changes easily, and generally has a resourceful nature.

To illustrate, an IS development project that closely follows the methodology of the waterfall model (Royce 1970) would be considered very non-agile, since the strictly sequential nature of that approach, in which coding does not start until design is finished, and design only starts after the completion of analysis, is not quick or adaptable at all: The whole system is finished only at the end of the project, and there is no mechanism to adapt to changes that are discovered during the course of the project – once for example the analysis phase is completed, requirements are supposed to be frozen. Near the other end of the agility spectrum, a project that closely follows the approach of Extreme Programming (Beck 1999) would be considered to be quite agile, since small parts of the whole system would be completed very quickly (at the end of each iteration). The iterative approach of XP also facilitates the incorporation of changes into the system, because requirements can be adjusted throughout the whole project.

In actual development practice, most projects fall somewhere in between these two extremes[3]. In other words, there is no dichotomy between plan-driven and agile methods, but a continuum of development methods (Ward and Legorreta 2009). Even in 1970, the author of (Royce 1970) realized that the waterfall method described in that paper is unrealistic in its pure form, and that there will always be the need to interact iteratively between the different phases. On the other

[2] http://oxforddictionaries.com/definition/agile?view=uk, accessed on Nov. 19, 2013
[3] Note that there are many different development methodologies that fall somewhere between the two extremes described here, such as the spiral model (Boehm 1988).

hand, many of the assumptions behind agile methods like Extreme Programming are idealistic as well. Turk, France et al. (2005) describe a number of implicit assumptions behind the agile manifesto, such as the assumption that a complex project can always be broken down into short iterations, that competent customer teams are continuously available to developers, and that the cost of changes to the system does not increase significantly over time. Since such assumptions will hold to different degrees in different projects, and due to other external factors imposed by real life (such as contractual or regulatory constraints), any specific development method like Extreme Programming will typically be adjusted for each individual project ((Henderson-Sellers and Serour 2005), (Aydin, Harmsen et al. 2005), (Fitzgerald, Hartnett et al. 2006)).

This "fuzziness" of the term of agility introduces an additional difficulty into empirical studies of the research questions above: How can the degree of agility of an IS development project be assessed? For example, if development practitioners are questioned about their conceptual modeling practices, how can it be determined whether they actually perform agile development and are thus of interest for this research?

Unfortunately, there is no consensus in the literature about the definition of the concept of agility in the context of IS development. Lee and Xia (2010) review the literature and list definitions from eight different papers (Conboy and Fitzgerald (2004), Highsmith (2004), Larman (2004), Erickson, Lyytinen et al. (2005), Henderson-Sellers and Serour (2005), Lyytinen and Rose (2006), Cockburn (2007) and Qumer (2008)). These definitions are, despite significant overlap, quite distinct and focus on different aspects of the concept, including "the continual readiness [...] to rapidly [...] embrace change", "the ability to both create and respond to change [...]", "[...] to strip away the heaviness in traditional software development [...]", "[...] the ability to fine-tune and reengineer software development processes when needed" and "the ability to sense and

respond swiftly to technical changes and new business opportunities [...]". Conboy (2009) examines the concept of agility in other fields such as manufacturing and management, and transfers its meaning to the IS development domain. He concludes by defining three requirements for agility: Flexibility, leanness and continual readiness (e.g., of intermediate work products).

1.2.2 Measuring Agility

Not surprisingly, since there is no generally accepted definition of agility, there is also no generally accepted measurement instrument for agility in the literature. In fact, there has been surprisingly little work on measuring the agility of IS development projects. Maruping, Venkatesh et al. (2009) developed their own measurement instrument for agile methodology use because they were not aware of any existing measures. Their instrument has three items each for six different agile practices drawn from Extreme Programming: Pair programming, continuous integration, refactoring, unit testing, collective ownership (of code) and coding standards. They justify this selection of a subset of all practices in Extreme Programming by arguing that these six practices "specifically promote flexibility and have been identified as instrumental in enabling software development teams to respond to requirements changes". This approach seems somewhat problematic, because it is specifically tailored towards Extreme Programming and might not adequately measure the agility of projects that use other agile methods. Furthermore, some of the six practices (for instance pair programming and coding standards) are not strictly tied to agile development, but might also be used as means for quality improvement in non-agile development.

Sidky and Arthur (2007) develop a more comprehensive instrument for assessing agility. They propose five levels of agility, and indentify about 40 specific agile practices that indicate

attainment of each level for each of the five agile principles defined in Beck, Beedle et al. (2001). Unfortunately, this approach suffers from the same limitation as the work by Maruping, Venkatesh et al. (2009) – it uses practices from specific existing agile methods, which makes it unreliable for the assessment of novel development methods. Furthermore, its complexity limits its usability to in-depth case studies and similar research methods, while a use in a survey would be prohibitive due to its size (approximately 300 items). The same has to be said about the approach taken by Kettunen (2012), who develops a set of about 500 items based on drivers, goals, means, enablers and impediments to agility. The purpose of that work is more on improving development processes in an organization than on measuring agility, and the author concludes that for that purpose, the "process of discussing and analyzing the various items proposed in the framework is often even more useful than the exact assessment outcome".

Qumer (2008) describes an analytical framework with which the degree of agility of different agile development methods can be compared. This framework, called 4-DAT, has four dimensions: Method scope (i.e., in which environments can the method be applied), features (for instance flexibility, speed, and leanness), agile values (the degree to which the values of the agile manifesto (Beck, Beedle et al. 2001) are respected) and process (degree to which development/project management/software configuration and process management processes are covered). Unfortunately this framework can only be used to analyze development methods abstractly. It is not broken down into a survey instrument that could be used to assess the level of agility of individual projects, and it is not clear how the framework could be operationalized for such a purpose.

Lappo and Andrew (2004) propose to use measurable goals instead of metrics to assess the level of agility in development projects. However, they only provide two example goals (frequent

delivery of working software and knowledge sharing) instead of a comprehensive list that could be used to measure agility.

Several approaches focus on development process improvement, and view the assessment or measurement of the degree of agility as an auxiliary task. For example, Packlick (2007) defines five high-level goals of agility, and proposes to assess the achievement of each of these goals at one of five maturity levels. However, the work does not contain specific ways to perform this assessment, is tied to practices and goals of a single specific agile development method (Extreme Programming), and is based on practical experiences from a single organization. The work by Buglione (2011) is similarly incomplete. The author proposes to use a light-weight version of existing maturity models to assess agility, and suggests four alternative sets of categories that could be used for this purpose, without discussing which one should be chosen, and how either of them could be measured in practice.

Patel and Ramachandran (2009) propose a software process improvement framework with five levels of agile adoption and maturity and assign agile practices to these levels. For example, the use of user stories on story cards for requirements purposes is argued to reflect level 2 ("Explored"), while self-organizing teams are put at level 4 ("Improved"). The usage of the practices is measured by approximately 20 items at each level. The main weaknesses of this approach are the reliance on specific agile practices, and the lack of justifications for the assignment of these practices to the five stages. Also, the large number of items (approximately 100 in total) is a concern.

Williams, Rubin et al. (2010) suggest to drive development process improvement by comparing the degree of agility in terms of seven dimensions (teamwork, requirements, planning, technical

practices, quality, culture and knowledge creation) against the industry standard (i.e. the average of other respondents). Similarly to other work, the size of the measurement instrument (125 items) is a concern, as is the lack of theoretical justifications for the seven dimensions and their measurements, and the inclusion of development-method specific practices such as pair programming or coding standards that are not necessarily indicative of agility.

A slightly different, but related approach is taken by Soundararajan, Arthur et al. (2012), who focus on how useful agile practices are in specific organizational contexts. The basis for their assessment are objectives, principles and practices of the agile paradigm. Their criteria are the adequacy of practices to support principles and objectives, the capability of the organization to use specific practices, and the effectiveness of these practices in the organization. Just as with most of the work discussed in this section, the reliance on specific agile development methods and their practices is one of the main drawbacks of this approach.

Finally, there are some non-academic attempts to measure agility. For example, a Swedish Consulting company developed an eleven-question survey to measure the agility of a given project (Seuffert 2009). This survey asks respondents for measurable project characteristics (such as the length of iterations) and their own assessment of important aspects such as cooperation across team boundaries. The obvious limitation of such practitioner-driven approaches is the lack of evidence for crucial aspects such as construct validity and reliability. Furthermore, no information about the theoretical justification of the specific survey questions is available. The same can be said about an instrument termed the Nokia Test[4] developed by Bas Vodde and Jeff Sutherland, which assesses how closely a development project follows the practices proscribed by the agile development method Scrum.

[4] http://jeffsutherland.com/nokiatest.pdf, accessed November 11, 2013

1.3 Domain Understanding and Conceptual Modeling

Wand and Weber (2002) give an overview of the literature on conceptual modeling in the IS field, and lay out a framework for a research agenda. They point out that the interest in the use of conceptual modeling in software development goes back to the 1960s (e.g. Naur and Randell (1969)), driven by the realization that insufficient understanding of requirements and the application domain were at least partially to blame for the prevalent failures of development projects. One of the most important modeling techniques still used today, the Entity-Relationship model, was introduced in the 1970s (Chen 1976).

Since then such a large body of work on conceptual modeling has been accumulated, and in particular so many different modeling techniques have been proposed, that the derogatory acronym "YAMA" – "yet another modeling approach" – is sometimes used to refer to the situation (Wyssusek and Zaha 2007). Siau and Rossi (2011) concur that there is a large number of modeling methods, but they point out that this is not necessarily a problem, but might simply be a result of the reasonable assumption that no single modeling approach can satisfy all situations. Specific contexts and purposes make specialized modeling techniques desirable. However, they criticize a lack of sound theoretical foundations behind most modeling approaches, and a scarcity of empirical evidence of their usefulness. Gemino and Wand (2004) share this view, and lay out a framework for the empirical evaluation of conceptual modeling techniques.

Because the agile paradigm has a bias against extensive documentation ("[...] value [...] working software over comprehensive documentation" - Beck, Beedle et al. (2001)), it is not surprising that agile methods do not typically prescribe conceptual modeling, and consequently that there is no literature about conceptual modeling in agile development. Instead, in order to achieve

domain understanding, agile methods rely on "constant collaboration between developers and users" (Rubin and Rubin 2010). As Kovitz (2003) points out, the difference between plan-driven approaches like the waterfall model and agile methods is that the former analyze the domain and the requirements early in the project and create domain models and requirements documentations, while the latter develop an understanding of the domain and the requirements throughout the project, typically without explicitly capturing either in documents.

Empirical studies confirm this approach in agile development practice. For instance, Cao and Ramesh (2008) study agile development projects in sixteen different organizations, and find that formal documentation of requirements is in fact not typically practiced, and that instead simple techniques such as user stories (brief descriptions of requirements on index cards, see Savolain, Kuusela et al. (2010)) are used. Practitioners' descriptions of agile development methods such as Scrum or Extreme Programming concur with this assessment. According to Rising and Janoff (2000), requirements in Scrum are kept in the so-called product backlog. The format of description of the requirements is not specified in detail, but the nature and use of the backlog (which is subject to frequent changes) implies a concise textual form. Similarly, Extreme Programming captures requirements in so-called story cards, which contain one requirement per index card (Beck 1999).

However, this status quo, in which agile development is equated to very limited if any explicit conceptual modeling and requirement engineering practices, is occasionally questioned. For example, Orr (2004) calls for more advanced requirements engineering practices in agile development, while Selic (2009) suggests to extend agile development methods by more extensive documentation in order to deal with high levels of complexity and to facilitate the

maintenance of long-lived systems. Even though the author mostly calls for architectural and design models, the same arguments can be made for models of the business domain.

In summary, while there is a large body of literature on domain understanding and conceptual modeling in general, there is very little work in these areas about agile development. This is not surprising, given how lean agile development practices typically are, especially related to requirements and documentation. Since the conceptual modeling literature contains little empirical work on strengths and weaknesses of specific modeling techniques, it is not clear which – if any – conceptual modeling methods might be suitable for agile development.

1.4 Thesis Outline and Contribution

This thesis is structured as follows: After this introduction, chapter 2 provides the theoretical background. It defines the concept of agility in a way that can be operationalized for a measurement instrument, and examines the impact that conceptual modeling should have on agility in IS development. It also outlines and justifies the research approach chosen.

Chapter 3 describes a survey of software developers about the level of agility of their projects, their conceptual modeling use, and issues they face with domain understanding. This survey serves to major purposes: Firstly, to validate a measurement scale of agility based on the definition of agility from chapter 2, and secondly, to collect data about conceptual modeling use in agile development practice, and about the impact of that conceptual modeling use.

Chapter 4 contains the second empirical study, an exploratory experiment to study the impact of conceptual modeling in a simulated agile development project. This study is informed by the results of the survey, for instance by focusing on measuring the aspects of agility that were found to be most important in the survey.

Chapter 5 summarizes the work and concludes with an outline of possible future work in this area.

This research makes several contributions to the literature and to practice:

- An operationalizable definition of agility in the context of IS development, along with a validated measurement instrument based on that definition
- A better understanding of current conceptual modeling practices in agile IS development
- Based on the experiment,
    - first insights into the mechanisms by which conceptual modeling practices might affect domain understanding and communication about the domain in agile IS development
    - a methodological contribution towards our understanding of how to run complex experiments that examine IS development in a more realistic way than is typically done in existing conceptual modeling research (in particular, including group tasks and the passing of time).

Chapter 2: Domain Understanding in Agile Information System Development

2.1 Introduction

This chapter explains the theoretical considerations behind the research described in this dissertation, and lays out the research approach in more detail. It begins by closely examining the concept of agility, and by positioning the applicability of this work in the wide range of settings that might be considered "agile". It continues by making the case for conceptual modeling in agile development, both from a theoretical perspective, and from existing empirical evidence. After that the research approach is described, first for getting more evidence about current practices in agile development, and then for evaluating what impact the proposed addition of conceptual modeling practices to current agile development approaches might have. Finally, as a foundation for these empirical studies, an operationalizable definition of agility is developed, and the case for conceptual modeling in agile development is revisited based on this definition.

2.2 Information Systems, Agile Development Methods and Agility

This work focuses on the role of conceptual modeling in agile information systems development, as opposed to agile software development in general. In the literature there is typically no clear distinction between the two, or even a definition of the former as a subset of the latter. A business dictionary[5] defines the term information system as "a combination of hardware, software, infrastructure and trained personnel organized to facilitate planning, control, coordination, and decision making in an organization." For the purposes of this work, an

[5] http://www.businessdictionary.com/definition/information-system.html, accessed on August 14, 2013

information system can thus be defined as a (typically complex) socio-technical system that is used in an organization[6].

An important part of this system will typically be implemented in the form of software, but the focus in information systems research is typically on its use by individuals in the context of an organization. In fact, IS research is often perceived as having neglected the actual (software) system itself (Weber 2003).

One important theoretical characterization is that information systems represent aspects of interest of the "real world". The phrase "real world" is not meant to imply a specific epistemological point of view. For our purposes, it does not matter whether one believes that there is an objective "real world" waiting to be discovered (positivism), or whether the "real world" is socially constructed (constructionism) (Guba and Lincoln 1994). The important aspect is that information systems represent important information of interest in an organization: For example, transactions recorded in an accounting system reflect actual transactions that took part in a company (Wand and Weber (1990), Recker, Rosemann et al. (2006)).

While the work presented here mostly also applies to software development in general (i.e., to software that would not be considered to be an information system, such as technical software packages), it is particularly relevant in the context of information systems, where successful system use at an individual, group and organizational level is crucial (Burton-Jones and Gallivan 2007). The organizational setting brings additional complexity such as conflicting goals or even conflicting understandings of the same thing by multiple stakeholders (Suchman 1995). Such difficulties occur less in other types of software systems.

[6] For different conceptualizations of information systems, please refer to (Orlikowski and Iacono 2001)

The close relationship between software systems in general and its subset of information systems allows us to build this work on both literatures. For instance, most work about agile development is on agile software development, rather than agile information system development. Yet, it still applies to information systems as well.

The literature views the concept of agility from two different perspectives: Either the term "agility" is treated as characterizing a set of similar, distinct software development methods such as Extreme Programming, Scrum, Feature Driven Development, Adaptive Software Development, Dynamic Software Development Method or Crystal (Qumer 2008), or the term refers to a (at least theoretically) observable property of software development methods, projects or organizations that could be defined for instance as "the quality or capability of being quick moving and nimble" (Lyytinen and Rose 2006).

Both views are clearly related – agile software development methods tend to deliver a higher degree of measurable agility than non-agile methods (Qumer 2008). Still, in order to remain relevant regardless of whichever specific agile development method (current or future) is used, this research focuses on the latter sense of the term agility, which describes a measurable and non-binary characteristic of projects, methods or organizations, and not the membership in a specific set of development methods.

2.3 The Case for Conceptual Modeling in Agile Development

2.3.1 Benefits of Conceptual Modeling in IS Development

Based on the definition of conceptual modeling as "the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication" (Mylopoulos 1992), a very general argument for the usefulness of conceptual

modeling can be made: Clearly, in any IS development project, proper understanding of and effective and efficient communication about the domain of the information system is useful. Therefore, at an appropriate level (at which these benefits are not outweighed by the costs of modeling), conceptual modeling should be useful in any IS development method.

A similar argument is made by Moody (2005), who lists prior work showing that more than half of the errors in system development are requirement errors, that such requirements errors are the most common reason why system development projects fail, and that fixing errors in the requirements stage is significantly cheaper than fixing them later in the development lifecycle. As a consequence, he suggests to improve the quality of requirements by improving domain understanding through conceptual models of higher quality.

As Nuseibeh and Easterbrook (2000) state, "only by describing the environment, and expressing what the new system must achieve in that environment, we can capture the system's purpose, and reason about whether a given design will meet that purpose." In other words, modeling the domain of a system, i.e. the environment in which a system will be used, is an important prerequisite for the successful determination of requirements, which in turn are of critical importance for overall project success. Thus, domain models should be a significant portion of the requirements engineering process. One of the key advantages of domain models is that they allow detailed reasoning about and validation of assumptions about the domain. According to Kung (1989) this benefit is not restricted to the early phases of a development project. Instead, conceptual models play an important role throughout the whole development lifecycle, from requirements analysis through design, implementation, test and maintenance.

Even though there is relatively little strong empirical data about the usefulness of conceptual modeling in systems development (Turk, Vijayasarathy et al. 2003), there is evidence that it is used widely. Davies, Green et al. (2006) survey approximately 300 development professionals in Australia, and find widespread use of conceptual modeling in development. Under the assumption of rational behaviour, this would imply that conceptual modeling is (or at least is perceived to be) useful. There are some studies, partially with limited external validity, that examine the value of conceptual modeling. An example in the context of database development is Turk, Vijayasarathy et al. (2003), who find in an experiment that the use of conceptual models improves the correctness of the resulting designs, and reduces the amount of time that subjects needed to perform the task.

Beyond the information system and software engineering literatures, other relevant fields such as education come to similar conclusions. For example, Mayer (1989) finds that the understanding of new material by science students can be improved by supplementing textual information with conceptual models. This finding might transfer to the context of IS development if there are similarities between the cognitive processes of a software developer or analyst learning about a domain in IS development, and the cognitive processes of a student learning about a new domain in school.

Conceptual models might be particularly relevant in the context of IS development, given that information systems can be viewed as representations of relevant aspects of the domain (Recker, Rosemann et al. 2006): If the information system under development is supposed to represent (relevant aspects of) the domain, then it seems reasonable to assume that representations (of relevant aspects) of the domain in the form of conceptual models would be useful to developers.

Finally, the interpretation of conceptual models as boundary objects suggests their usefulness in IS development. A boundary object can be defined as "an artifact or a concept with enough structure to support activities within separate social words, and enough elasticity to cut across multiple social worlds" (Bergman, Lyytinen et al. 2007). In information system development, the different stakeholders involved, such as developers, users, and managers, represent these different "social worlds", because they typically have significantly different backgrounds and experiences, which results in difficulties to communicate effectively and efficiently between different groups. Conceptual models can serve as boundary objects to bridge these boundaries, and help to overcome some of the difficulties arising from the involvement of different stakeholders, such as multiple interpretations of the same concepts or processes (Suchman 1995).

2.3.2 Potential Benefits of Conceptual Modeling in Agile Development

In light of the bias against extensive documentation in the agile paradigm, as described in the "agile manifesto" with the words "[...] we have come to value [...] working software over comprehensive documentation" (Beck, Beedle et al. 2001), it is not surprising that agile development methods typically do not prescribe significant documentation practices. Rumpe (2002) explains that the negative view on documentation in agile development is motivated by the desire to reduce the workload of developers, and by the observation “that developers don’t trust documents, because they are usually out of date”.

However, it is worthwhile do differentiate between different kinds of documentation used in development. While documentation about artifacts that are subject to frequently changes in agile development, such as source code and to a lesser degree software design, requires significant amounts of work to keep up-to-date and thus useful, documentation about less volatile aspects of the project might be more practical. In particular, knowledge about the domain of an information

system should be much more stable, since the domain can be expected to change less frequently than the source code of the system under construction. Therefore, the use of conceptual models (i.e., models of the domain) might be feasible even in agile development.

Whether or not the use of conceptual modeling would in fact be beneficial in agile development is a question that can ultimately only be answered empirically. Essentially, two opposing goals compete against each other – the need to be lean and reduce overhead during development, and the need to understand the domain well and communicate about it effectively. The creation of conceptual models early in a development project might reduce leanness and increase overhead at least somewhat, an investment that might or might not pay off throughout the project.

It is important to emphasize that the argument is not that non-agile development practices are superior to agile practices. Instead, the argument is that it might be possible to carefully extend agile development approaches by specific conceptual modeling practices without significantly reducing their level of agility, and that doing so could lead to improved project outcomes. In other words, simply copying prescribed conceptual modeling practices from non-agile development approaches is not suitable, because it would likely render the resulting development approaches too non-agile. For example, if an agile project were to create a complete conceptual model of the domain at the beginning of the project (like the waterfall method would suggest (Royce 1970)), it would significantly reduce the level of agility of the project.

In addition to the points about the usefulness of conceptual modeling in IS development in general, an argument can be made that conceptual modeling could be particularly useful in agile development. As Rubin and Rubin (2010) and Abrahamsson, Warsta et al. (2003) point out, the scarcity of documentation in agile development results in the need for increased communication

between developers and users, and an increased importance of knowledge transfer, both during development, with practices such as pair programming with frequently changing pairs (Dawande, Johar et al. (2008), Balijepally, Mahapatra et al. (2009)), and during the maintenance phase of completed systems. Conceptual models can play an important role in facilitating such communication. In addition to that, conceptual models can be used to help keep requirements consistent, which can be particularly challenging in agile development, where requirements are defined incrementally throughout the project life cycle (Liu, Wang et al. 2010).

2.3.3 Existing Empirical Evidence

In addition to the theoretical arguments for the potential usefulness of conceptual modeling in agile development, previous studies found empirical evidence. From case studies in five organizations using agile development, Hansson, Dittrich et al. (2006) conclude that "documentation is in many cases a topic that leads to the discussion of the short-comings of today's practices", and that different projects come up with different solutions for this issue.

Cao and Ramesh (2008) study sixteen different organizations that use agile development and find that replacing written specifications by face-to-face communication often creates challenges. For example, most projects have difficulties meeting the assumptions behind many agile development methods, such as the continuous availability of knowledgeable users to the development team. In such situations, any documentation about the domain in the form of conceptual models might prove useful. Furthermore, the study finds that the scarcity of documentation related to requirements often leads to significant difficulties when personnel turnover occurs or when the complexity of the application grows. This can lead to "inability to scale the software, evolve the application over time, and induct new members into the development team". Again, conceptual modeling practices might alleviate such difficulties.

Based on the same case studies, Ramesh, Cao et al. (2007) find that current agile development methods lack sufficient practices to check the consistency of requirements. Conceptual models could play an important role in this context as well, by providing a basic vocabulary and framework against which incrementally defined requirements can be evaluated.

Finally, Conboy, Coyle et al. (2011) use case studies in seventeen organizations to investigate "people challenges" that arise from the adoption of agile development methods. One of the key challenges the study finds is a lack of business knowledge among developers. While agile development does not cause insufficient business knowledge in developers, it exposes it much more than other development methods would, because of the increased amount of direct face-to-face communication between developers and users. The consequences of this include not only inefficient and ineffective communication, but also a lack of trust between users and developers (especially since the former might lose confidence in the competence of the latter). The creation of conceptual models of the domain could contribute significantly towards restoring that trust and confidence, and towards more efficient and effective communication.

2.4 Research Approach

As indicated in the introduction, the goal of this research is twofold: Firstly, to learn more about the actual (as opposed to prescribed) conceptual modeling practices that are used in agile IS development, and secondly, to investigate the potential usefulness of conceptual modeling practices in agile development.

For both goals different research approaches are possible. Dennis (2001) points out that there is no perfect research method, but that instead each specific method has its own strengths and weaknesses. Based on McGrath (1981), he lists three dimensions by which alternative research

methods can be evaluated or compared: Generalizability with respect to the population of interest (also referred to as external validity), realism of the study for the participants, and the degree of precision with which the variables of interest can be controlled or measured.

Different research methods satisfy the three dimensions to varying degrees. For example, while laboratory experiments allow for a lot of control, they are weaker with respect to generalizability and realism. In contrast, field studies maximize realism, but satisfy control and generalizability to a lesser degree (Dennis 2001). The obvious conclusion is that different research methods should be combined (Mingers 2001), either within a single study, or in different studies in the same topic area. Unfortunately, this is still relatively rare in the IS literature (Mingers 2003).

Specifically for research in IS development and software engineering, Wohlin, Hörst et al. (2003) list four major types of empirical research approaches: Experiments, case studies, surveys and post-mortem analyses. Wynekoop and Russo (1997) classify research in IS development in a somewhat broader and more differentiated way, and list the following nine different research approaches: Normative writings, laboratory research, surveys, field inquiries, case studies, action research, interpretive research, descriptive research and practice descriptions.

Based on the considerations above, the following sections discuss the advantages and disadvantages of various alternative empirical research approaches for the two main research goals, and describe and justify the approaches chosen.

2.4.1 Understanding Existing Development Practices

Several different research approaches are possible to find out more about actual conceptual modeling practices in agile development. The potentially most obvious one – a review of the relevant academic literature – alone is insufficient, because the literature about agile

development is still immature and scarce in strong empirical studies (Dyba and Dingsoyr 2008). Of the existing work, very little is concerned with conceptual modeling, which is not surprising, given the general bias against documentation and modeling in agile development (Beck, Beedle et al. (2001), Rubin and Rubin (2010)).

Therefore, it is necessary to generate empirical data from primary sources, i.e., from IS development practice. Two main approaches for this seem practical: An in-depth focus on a small number of agile development projects in case studies (Benbasat, Goldstein et al. 1987), or a survey of a larger number of practitioners which would necessarily be less deep. Both approaches have their respective advantages and disadvantages, and both will hopefully be pursued by different researchers in the future, as they could clearly be complementary in the spirit of multi-method research (Mingers 2001). However, for practical purposes, one approach had to be picked in the context of this dissertation.

The selected approach is that of an online survey of practitioners. The main reason for this choice is that an online survey allows for the most generalizability of the results, because it can cover a much larger number of different projects. Case studies would require significantly more time, and would thus for practical purposes be limited to a small number of projects, increasing the risk that atypical projects were chosen by chance. Furthermore, because this is only the first of two steps of this research, one of the key benefits of case studies, the potentially bigger depth of insights gained, is less critical. Similarly, since the topic of investigation is not completely new – there is a body of literature in both agile development and conceptual modeling – enough knowledge exists about the phenomena of interest to formulate meaningful survey questions. Again, if that was not the case, case studies might have been preferable, because they would

allow to study completely unknown phenomena, for example to build theory from the bottom up (Eisenhardt and Graebner 2007).

The choice of a survey as the research approach has another significant benefit: It allows for a sufficient number of responses to meaningfully test a practical measurement instrument for agility. As discussed in the introduction, an operationalizable definition of agility and a measurement instrument based on that definition are important side products of the research questions posed, because it is impossible to determine actual conceptual modeling use in agile development without adequately determining or measuring agility.

2.4.2 Investigating the Impact of Conceptual Modeling

As suggested by Hevner, March et al. (2004), there is a wide range of possible approaches to evaluate IT artefacts such as development methods or practices. In the context of this work, the goal of the evaluation is to see whether extending a typical agile development method by suitable conceptual modeling practices is beneficial. Even though some analytical arguments about this are possible, it is ultimately an empirical question that requires an empirical investigation. Different empirical research approaches are possible, ranging from controlled laboratory experiments to observational field studies. Similar arguments as above apply again – each of these approaches has its specific strengths and weaknesses. For example, a laboratory experiment in which a treatment group performs agile development tasks using conceptual modeling practices, while a control group performs the same tasks without conceptual modeling would give the researcher a high degree of control over all variables, while the degree of realism of such an experiment would be limited. On the other hand, a field study in which the researcher observes different agile development teams, some of which use conceptual modeling practices and some of which do not, would be much more realistic, but it would be extremely difficult to

control all variables of interest involved, for instance the degree to which conceptual models are actually used.

After careful consideration of this trade-off, the selected approach is to use an exploratory laboratory experiment as a part of this research, and to postpone an evaluation in the field to future work. This choice is primarily based on the conviction that at this early stage of research in conceptual modeling in agile development, strong control over the variables of interest is more important than a high degree of realism. An additional reason is the fact that field studies would have brought significant risks, in particular in the recruitment of organizations willing to participate (and adjust their development methods based on outside suggestions of the researcher by adding conceptual modeling practices), and the ability to retain cooperation by these organizations. Finally, it should be noted that the first empirical part of this research – the survey about current practices – is performed in the field. Therefore, executing the second empirical part in the laboratory seems a suitable way to balance the advantages and disadvantages of both research settings.

2.5 Defining and Measuring Agility

2.5.1 The Need for an Operationalizable Definition of Agility

In general, it seems prudent to properly define major concepts related to ones research. In addition to that, the creation of a survey about conceptual modeling in agile development makes the need of a clear definition even more pressing, because a clear understanding of the concepts the survey questions are based on by the respondents of the survey is an obvious prerequisite for meaningful answers. As discussed in section 1.3, the definition of conceptual modeling by Mylopoulos (1992) as "the activity of formally describing some aspects of the physical and

social world around us for the purposes of understanding and communication" is well accepted and specific enough.

However, as outlined in section 1.2.1, the precise meaning of agility in IS development is much less clear. Yet, in order to be able to learn about current conceptual modeling practices in agile development, it is necessary to determine which respondents of the survey actually perform agile development. Therefore, the survey has to contain questions that measure agility. Creating such questions is more difficult than it might seem. A simple yes/no question, such as “Did you use an agile development method?” would be inappropriate, because agility is not a binary property that is either present or not. Instead, there is a range of IS development methods and variants from extremely rigid or plan driven to extremely agile. As Boehm (2002) points out, there is no dichotomy of agile versus non-agile development methods, but a spectrum of methods from completely unstructured approaches to very rigid, plan-based ones.

A further complication stems from the fact that organizations tend to adjust and fine-tune development methods to their specific needs (Aydin, Harmsen et al. 2005). Not only is this often necessary to satisfy specific organizational or regulatory needs, but their own modification is actually encouraged by many agile development methods. For example, Beck (1999) explicitly asks development teams to “change the rules at any time” in Extreme Programming. Therefore, different development teams using (adaptations of) the same development method might still end up with varying degrees of agility. This makes the assignment of a general agility score to specific methods inaccurate, and prohibits the inference of the degree of agility of the respondent from a question such as "Which development method are you using?".

Finally, even a self-reported agility score, with a question such as “In a range from 1 to 7, with 1 being completely plan-driven and 7 extremely agile, how would you rate the agility of your project?” would be very problematic, because the assessment would be very subjective in the absence of a clear standard of comparison. For instance, a moderately agile project would probably be assessed as quite agile by someone with an otherwise predominantly plan-driven background, while it might be rated as fairly non-agile by another developer who had previously only worked in very agile settings.

Therefore, a set of questions based on a suitable definition of agility is desirable to measure the degree of agility of the respondents' projects. Unfortunately, most definitions of agility in the literature describe the goal state of agility, and not the means by which it can be achieved. For example, Conboy (2009) defines agility as a combination of flexibility, leanness and continual readiness (of intermediate work results). While this is a perfectly fine definition to capture the concept of agility, it is less than perfect to measure it. In particular, flexibility and leanness are subjective and relative terms, and it seems difficult to derive short survey questions that can measure them in a reliable and valid way. For this reason, the following section provides a more operationalizable definition of agility.

2.5.2 Key Dimensions of Agility

To determine the defining dimensions of agility, it is useful to recall the goals behind the agile paradigm. As Boehm (2006) points out, one of the key driving forces behind the development of agile development approaches was the increasing rate of change, caused by ever more rapid technological innovation cycles, the emergence of the Internet and mobile devices, and the ever growing importance of time-to-market. Furthermore, even if the environment in which an information system is created were to be completely stable for the whole duration of the

development project, it would still be impossible to perfectly correctly define all requirements in the beginning. Royce (1970), who describes the waterfall model, recognizes that errors or omissions in the analysis phase will invariably be found later in the project, for example during design, implementation or test. This is not only caused by the fact that humans are fallible, but also a result of the enormous difficulty of describing a system completely in the abstract. Similarly, Ralph (2010) shows empirically that developers are in principle unable to design a system in the abstract, and that instead the understanding of the problem and its solution co-evolve throughout the construction of the system.

Based on this, it can be argued that the key objective of agile development methods is to respond to inevitable change as effectively and efficiently as possible while minimizing risks. In other words, instead of attempting to prevent the occurrence of the need for changes by an ever increasing "process bureaucracy" (Boehm 2006), the frequent need for changes is accepted, and efforts are concentrated on dealing with changes effectively (i.e., recognizing the need for changes to requirements and incorporating them in the system as quickly and correctly as possible) and efficiently (i.e., minimizing the amount of resources needed to do so).

In order to achieve this, the first step is that the need for a change (e.g., the change of a requirement or the rise of a new, previously unknown requirement) is discovered as early as possible. Clearly, if it is not recognized that a change needs to occur, it is also impossible to quickly (effectively) act on the required change. In order to be able to recognize the need for changes early, two additional conditions have to be met: The development artefacts must have a high degree of tangibility, and the project must respond to previously discovered required changes rapidly.

A high degree of tangibility means that the artifacts under construction can be easily inspected and observed throughout the project, typically by future users of the system. In agile development, this requirement is usually met by practices such as continuous integration (Lindvall 2004) in the form of a "nightly build", in which the latest version of the software is compiled at least on a daily basis, resulting in a usable – albeit only partially finished – system. This makes the system under construction very tangible – future users can experience the software directly by using it (either for real or simulated work tasks). Compared to less tangible representations of the work in progress, such as textual or graphical descriptions of requirements or the user interface, a working piece of software makes it much easier for users to discover errors, misunderstandings, omissions or other issues that need to be fixed and thus create the need for changes.

It should be noted that tangibility is not a binary property that is either present or not, but a continuum. For example, working (prototype) software would be highly tangible, written requirements in design documents highly intangible, while other mechanism such as screen mock-ups could be rated as having moderate tangibility, because they show users what the finished application might look like, without allowing them to actually experience the behaviour of the software.

A quick response to recognized required changes is the other main prerequisite for a quick discovery of the need for changes, because only a reasonably current early version of the system allows users to find additional errors, misunderstandings or omissions. Frequently changes lead to the need for additional changes. A typical example would be an error in the software that is recognized by a user, but because of a misunderstanding in the communication with a developer the solution is not completely understood by the developer. In addition, if users request changes

to the software, but these changes are not incorporated into the software within a short timeframe, the motivation of users might suffer, changes might be requested repeatedly (by different users, or simply by users who forgot that they already requested a specific change), and conflicting descriptions of required changes might arise.

In order to achieve a quick response to changes, another important condition has to be met – low overhead. This is essentially equivalent to the leanness condition of (Conboy 2009), and is reflected in the agile paradigm by the reduction of documentation, which is often considered superfluous (Beck, Beedle et al. 2001), and by the insistence that developers should "implement only the functionality demanded by the stories in this iteration" (Beck 1999), and not add any additional features that they assume might be needed later. The key benefit of low overhead (besides lowering the overall effort spent and potentially wasted) is that it keeps the project nimble, and enables developers to keep up with changes in a timely manner as they arise.

The final key dimension of agility, process agility, is orthogonal to the ones discussed so far. A development team possesses process agility if it continuously adjusts its development process to changing environments and constraints. Software process improvements are not limited to agile development, but are performed by many software development organizations (Iversen, Mathiassen et al. 2004). In a common framework of process maturity (Humphrey 1988), the consistent optimization of development processes is the characterizing factor of the highest level of process maturity.

In contrast to this, process improvement in agile development approaches is typically performed within a much shorter timeframe and in a much more ad-hoc nature. As Talby, Hazzan et al. (2006) point out, typical agile projects might perform a so-called reflection meeting at the end of

each iteration, i.e., for instance every two weeks. In such meetings, the last iteration of development is reviewed, and based on problems or issues encountered, modifications to the development practices for the next iteration might be made. Depending on the nature of the changes to the development process, the effectiveness or efficiency of the team and all the other dimensions of agility might be improved.

In summary, the key dimensions of agility are as follows:

1. Early recognition of the need for changes
2. High degree of tangibility of intermediate results
3. Quick response to recognized required changes
4. Low overhead/leanness
5. Process agility

The key dimensions are strongly interdependent. Process agility supports all others, because it allows to correct weaknesses in the development process that could affect either one of the other dimensions. Low overhead primarily supports the ability to respond quickly, while a quick response to recognized changes and a high degree of tangibility are crucial for early recognition of the need for changes.

Finally, it should be pointed out that all these dimensions are much more measurable and less subjective than many other characterizations of agility such as flexibility. For instance, the tangibility of intermediate results can be rated based on the degree to which practices such as the early delivery of software to users or mock-up screens or prototypes are used. Details of the measures used in the survey will follow in the next chapter.

2.5.3 The Case for Conceptual Modeling in Agile Development Revisited

Given the operationalizable definition of agility as the five key dimensions, the question whether conceptual modeling practices can potentially improve agile development can be revisited. In order to do this, we compare three different alternative conceptual modeling strategies for agile development: the complete omission of conceptual modeling, conceptual modeling as practiced in non-agile methods, and the adaptation of traditional conceptual modeling practices to agile development approaches.

In a project that performs no explicit conceptual modeling, the early recognition of the need for changes might be negatively affected, because the communication between users and developers could be less effective and the domain understanding by developers might be reduced. Therefore, discovering for instance misunderstandings between developers and users would be harder, potentially delaying the recognition of errors. Similarly, since conceptual models themselves can serve as reasonably tangible early work results, not having them at all would reduce tangibility. Furthermore, the quickness of responses to recognized errors could be compromised without conceptual models, due to the more difficult and less efficient and effective communication between developers and users about found errors. While omitting conceptual modeling practices altogether might be viewed as a reduction of overhead (since the task of creating conceptual models is eliminated), it could at the same time possibly cause significant overhead, for instance when users have to repeatedly explain the same concepts to different developers, in the absence of conceptual models that the developers could use instead. Selic (2009) argues similarly that adding documentation practices to agile development might reduce costs in the maintenance phase of projects. Finally, process agility would suggest that conceptual modeling practices can

and should be added to a project if they are beneficial – thus, a blanket rejection of conceptual modeling would clearly violate this dimension of agility.

Alternatively, an agile project could attempt to use conceptual modeling as it is performed in non-agile development approaches like the waterfall model, in which complete conceptual models of the relevant aspects of the domain are created in the initial requirements and analysis phases. Such an approach would also violate the key dimensions of agility. The early recognition of the need for changes would be hampered by spending a significant amount of time in the beginning of the project in which only analysis, but no design, coding or testing occur. Work results with a higher degree of tangibility (in particular, working software) would be delayed accordingly, and if needed changes were recognized early, then they could only be implemented in software later in the project. Modeling the complete domain up-front would also create significant overhead, because unnecessary parts of the domain might be modeled (due to incomplete understanding of actual user requirements at the time), and because many changes to the models would probably be necessary throughout the whole project. Again, process agility would be violated by a fixed prescribed conceptual modeling approach.

In summary, both the complete omission of conceptual modeling and the execution of conceptual modeling based on non-agile development practices negatively impact the agility of software development. This raises the possibility that a conceptual modeling approach that is adjusted to agile development might be a better solution.

Such an agile conceptual modeling approach could follow the suggestion by Beck (1999) by performing a brief initial period in the project in which a high-level overview of the domain is created, with the intention to come up with a common language about the key entities in the

domain, and to delineate the boundaries of the relevant domain as understood at this point. In contrast to the description in Royce (1970), this initial phase would not have a duration in the order of magnitude of months, but only last a number of days or at most a few weeks. The initial conceptual model would then be extended incrementally, by adding whatever relevant information about the domain is uncovered during the implementation of different requirements throughout the project.

The initial model creation could be based on the initial set of user stories that are created in the first iteration of the project (see Beck (1999)). Depending on what type of models are found to be useful in a specific setting, various types of models could be derived from this information. For example, a use case diagram depicting the roles (extracted from the user stories) and the use cases (the user stories themselves) these roles can invoke. Another possibility might be activity or process diagrams depicting the dynamic aspects of the domain, or class or entity-relationship diagrams about the various concepts of interest and their relationships. The most important aspect is that irrespective of the types of models created initially, all models would only be considered a snapshot of the current understanding of the domain, and expected to be extended and corrected as the understanding of the domain evolves and grows.

Such a modeling approach would be very compatible with the five dimensions of agility as defined above. In particular, it would facilitate the early recognition of the need for changes, because it would on the one hand not significantly delay the creation early versions of the system, and on the other hand support the discovery of misunderstandings and other types of errors by providing a common language and understanding between developers and users.

Tangibility should not suffer for the same reason, since more tangible work products like early system versions would not be delayed significantly. The loss of tangibility due to the time spent on creating conceptual models might at least partially be compensated for by the comparatively tangible nature of conceptual models themselves. By giving users or other testers a bigger context in which the system under construction can be evaluated, conceptual models could improve the usefulness of early system versions.

The proposed modeling approach could also increase the ability of developers to respond quickly to recognized required changes, because the models could increase domain understanding of developers and improve communication between developers and users, making the implementation of changes more efficient and effective.

The exact same argument could be made for the reduction of overhead – a better understanding of the domain and improved communication would reduce the extra work required due to continued misunderstandings, or simply because developers could gain required knowledge from the existing models as opposed to having to repeatedly retrieve it through inquiries with users. Vidgen and Wang (2009) use case studies to investigate enabling and inhibiting factors to agility in software development, and find that over-communication between developers and customers and between development team members are inhibitors to agility. Conceptual models might help reduce such over-communication and the overhead it causes. These benefits might more than compensate for the increased effort of creating and updating the conceptual models throughout the project, provided that the tools used allow this to be performed efficiently and that the domain is at least somewhat stable.

Finally, process agility would not be affected negatively either, because the details of the modeling approach could be adjusted as needed throughout the project.

In summary, the proposed operationalizable definition of agility in the form of five key dimensions supports the hypothesis that agile development practices can be improved by appropriate conceptual modeling practices.

Chapter 3: An Examination of Current Agile IS Development Practices

This chapter describes the survey about actual conceptual modeling practices and domain understanding in agile development. It begins by explaining the survey design, followed by details about the execution of the survey, and concludes by a detailed analysis of the results.

3.1 Survey Design

Even though the main objective of the survey is to find out more about conceptual modeling practices and domain understanding in agile IS development, it has to be designed in such a way that it can be answered meaningfully by developers using any development approach, agile or not. There are two main reasons for this: Firstly, as discussed above, "agility" is a poorly understood and vague concept which is not binary in nature, but exists in a continuum from "extremely agile" to "not agile at all". Therefore, it is impossible to target it exclusively at developers using agile methods. Instead, respondents will reflect varying degrees of agility, and the survey should be sensible to all of them.

Secondly, since our knowledge of conceptual modeling and domain understanding in practice is limited in general, results exclusively about agile developers would be hard to interpret. Instead, it is much more useful to be able to contrast practices between practitioners who use fairly agile approaches, and ones who use more traditional, plan-driven approaches. For example, information about the types of conceptual models that are used by agile developers is much more meaningful if can be compared to the types used by less agile developers.

3.1.1 Construct Validity[7]

The individual items of the survey were developed based on a detailed study of the relevant literature with existing measurement instruments and about agile IS development methods and practices. Their clarity and understandability was improved through various rounds of pre-pilot tests of the measurement instrument with researchers (e.g., in software engineering and information systems) and practitioners familiar with agile development.

With any measurement instrument, ensuring validity is a major concern. Construct validity can be defined as "the extent to which an operationalization measures the concept it is supposed to measure" (Bagozzi, Yi et al. 1991). Threats to construct validity include the possibility that respondents understand questions differently than intended, and that questions earlier in the survey influence answers to questions later in the survey.

One of the key measures taken to increase the confidence in construct validity is that there are multiple items for each construct. This reduces measurement error, since any individual question is likely to be less reliable than a combination of related questions (Churchill 1979). If different items for the same construct correlate very poorly, it might be an indication of an unclear item[8].

Furthermore, most questions are designed to gather information of a more factual nature than about subjective perceptions. Even though the responses are of course still only perceptions of respondents about facts, this nature of the questions should still increase confidence in the answers. For example, instead of simply asking subjects on a Likert-scale about how quickly they can discover the need for changes in their projects, the questions are about more factual information, such as the frequency with which they give early version of the software to end

[7] This section discusses threats to construct validity. External validity is examined in section 3.2.3. Internal validity does not apply, because the survey does not attempt to establish the causal relationship between different variables.
[8] It turned out that the items correlated fairly well. For details, please refer to section 3.3.3.

users and about the ways in which end users communicate with developers about requested changes.

This type of item is quite different from other reflective measurement items that are typically used in the IS literature, because they do not measure attitudes, perceptions or intentions, but (perceptions of) related yet separate facts. This makes some of the measurement validation approaches as recommended in the literature, for example the technical validation of construct validity as described in Straub (1989), not applicable.

To illustrate this point, if a psychological variable such as "behavioral intention to use the system" is measured by the items "I intend to use the system in the next <n> months", "I predict I would use the system in the next <n> months" and "I plan to use the system in the next <n> months" (Venkatesh, Morris et al. 2003), anything but a very strong correlation between these three items would have to be reason for concern. In contrast to this, to measure the early recognition of the need for changes, different items measure how often (if ever) users are provided with early releases of the system under construction for testing purposes, and how often (if ever) users receive early releases for their daily productive work. While both items reflect the ability to recognize the need for changes early, they cannot be expected to correlate perfectly, because using early releases for only one of the two purposes is a perfectly reasonable development approach.

Another specific threat with regard to items about conceptual modeling lies in the fact that respondents might include design models of a conceptual nature in the category of conceptual models. To mitigate that risk, all the questions about models are explicitly asked pair-wise about

both types of models. This approach should keep respondents from forgetting about the distinction.

Finally, the order of questions was chosen to minimize the risk that earlier questions influence the answers to later questions. In particular, the questions about modeling use come after the questions measuring agility. Thus, the answers of respondents to the questions measuring agility are not influenced by the fact that they are prompted to think about models by the questions about modeling. In the reverse order, this could have been an issue for instance for respondents who typically do not use models. Of course, the order of questions could still influence answers. For example, the measurement instrument of agility and the questions about modeling might influence the answer to the later question about perceived agility. However, the questions have to be asked in some order, and the expected achievable sample size made a randomization of question order (paired with statistical analysis of whether and how the order of questions influences responses) impractical.

The following sections describe the final structure of the survey after several rounds of refinements (for details about the pilot tests see 3.2.2): Measurements for agility, conceptual modeling practices, domain understanding issues, and demographic information. Unless specified otherwise, respondents are asked to answer questions about the current project on which they spend the majority of their time on (or on their last project, in case they are not currently on a project).

3.1.2 Measuring Agility

As described in section 1.2.2, existing measurements for agility are not usable in the context of this survey, because they are either tied too strongly to one specific agile development method

like Extreme Programming (e.g., Maruping, Venkatesh et al. (2009)), because they are designed to measure the level of agility of a development method instead of a development project (e.g., Qumer (2008)), or because they are not developed sufficiently to measure the whole construct of agility (e.g., Lappo and Andrew (2004)).

Therefore, this survey uses a measurement instrument for agility based on the key dimensions of agility as described in section 2.5.2. The complete measure itself can be considered a formative (as opposed to a reflective) construct (Diamantopoulos and Siguaw 2006) – the five key dimensions all contribute to agility, without necessarily being strongly related. For example, a specific project might be using highly tangible intermediate work results, yet be unable to quickly respond to recognized changes due to an insufficient number of developers. However, this does not imply that the dimensions are completely independent – as discussed above, they are in fact interdependent, for instance because a higher degree of tangibility would arguably improve the chance to discover required changes quickly.

In order to be applicable to all software development practitioners, independently of whether they use an agile development method or not, none of the questions refer to or assume any specific agile development practices. For example, there is no question about the length of development iterations, because that would assume iterative development with fixed length iterations. In addition to this, subjects are instructed at the beginning of the survey that all questions are optional, and that they should omit any questions that seem not applicable to their project.

The following sections show the final measures for each of the five key dimensions.

3.1.2.1 Early Recognition of the Need for Changes

```
1. Do you provide early versions of the software to end users so that they can try it
out?
       ( ) Never
       ( ) Seldom
       ( ) Sometimes
       ( ) Often
       ( ) All the time

2. If you do provide users with early versions of the software, how often do you
typically do so?
Every...
       days _________________
       weeks ________________
       months _______________

3. Do you release early versions of the software to end users to use in their daily
work?
       ( ) Never
       ( ) Seldom
       ( ) Sometimes
       ( ) Often
       ( ) All the time

4. If you do release early versions of the software to end users to use in their daily
work, how often do you typically do so?
Every...
       days _________________
       weeks ________________
       months _______________

5. How do end users give you feedback on the software or request changes to it? Check
all that apply.
       ( ) Through a formal change request process
       ( ) Users approach developers with feedback
       ( ) Developers approach users for feedback
       ( ) Regular meetings
       ( ) Other (please specify):
```

Questions 1 and 2 ask about the frequency with which end users are provided with early trial versions of the software, while questions 3 and 4 ask about the frequency with which early releases of the software for productive use are provided. Both are indicators of how well the need for changes can be discovered in a timely manner, because the use of early versions of the system by users is key to the recognition of errors or omissions. While developers can recognize the need for certain changes without user involvement, changes that are triggered by or require detailed knowledge of the business can often only be detected by end users. Question 5 asks

about how discovered errors and requested changes are communicated from users to developers, because the need for a change is – from a project perspective – only truly discovered once the initial discovery by a user is communicated to the development team.

3.1.2.2 High Degree of Tangibility of Intermediate Results

```
1. What approaches do you use to give users a preview of what the finished software
will look like?
                          All the time  Frequently   Sometimes    Rarely       Never
Mock-ups (e.g. of screens) _____        _____        _____        _____        _____
Prototypes                 _____        _____        _____        _____        _____
Early, incomplete software _____        _____        _____        _____        _____
Other                      _____        _____        _____        _____        _____

2. If you indicated "other" in the previous question, please list them.
      _____________________________________________
      _____________________________________________
      _____________________________________________
      _____________________________________________
      _____________________________________________

3. What percentage of new or modified features would you estimate users actually try
out in early releases?
      _____________________________________________
```

Questions 1 and 2 measure which kinds of tangible intermediate results are used, and how often they are used. Question 3 is an estimate of the percentage of features that users actually try out in early releases of the software. The higher this percentage, the higher the tangibility in the project, because actual thorough testing and/or use of early releases of the system is just as important as the provision of such releases.

3.1.2.3 Quick Response to Recognized Changes

```
1. On average, how much time elapses between when the need for a critical change in
the software is identified and when the team starts working on it?
      days _________________
      weeks ________________
      months _______________

2. On average, how long does it take to complete such a critical change?
      days _________________
      weeks ________________
      months _______________
```

```
3. What percentage of all change requests would approximately fall into this critical
category?
      _____________________________________________

4. In your current or most recent project, consider the backlog of all changes not
deemed to be critical that have been requested by users. Assuming no new changes, how
long would it approximately take to incorporate all change requests (if no other
development would occur at the same time)?
      days _________________
      weeks _________________
      months _________________
```

Questions 1 and 2 measure how long it takes to complete critical changes to the software. The response time to critical changes is more useful to evaluate than the response time to all requested changes, because the latter might include requests for slight improvements or for additions of new functionality that are purposely (and without significant negative side effects) postponed for extended periods of time. This might especially happen in long running or larger projects, and measuring it would not be meaningful to assess agility. Question 3 is about the percentage of change requests that are considered to be critical. A very high percentage might indicate problems with the prioritization of change requests. Question 4 measures the size of the backlog of all requested changes. It has to be evaluated in relation to the overall duration of the project – a one-month backlog in a twelve-month project would indicate a generally much faster response to requested changes than a one-month backlog in a two-month project.

3.1.2.4 Low Overhead/Leanness

```
1. What percentage of your time do you approximately spend on each of the following
tasks? (Percentages should add up to 100%)
      Understanding requirements in meetings with users    _________________
      Understanding requirements (without users present)   _________________
      Design                                               _________________
      Coding                                               _________________
      Testing                                              _________________
      Other                                                _________________

2. What percentage of functional features that you implement is not directly derived
from user requirements, but added because they are likely going to be needed anyways?
      _____________________________________________
```

```
3. What percentage of non-functional features that you implement is not directly
derived from user requirements, but added because they are likely going to be needed
anyways?
      _____________________________________________

4. To what degree can you trace from requirements to the final code, in other words:
To what extent can you directly and easily determine for a given piece of code which
user requirement(s) it is based on?
      ( ) Never
      ( ) Seldom
      ( ) Sometimes
      ( ) Often
      ( ) All the time

5. What percentage of your work is re-work due to prior misunderstandings, inaccurate
requirements etc.?
      _____________________________________________
```

Question 1 directly measures (the perception of) the proportion of time developers spend on various kinds of tasks. Questions 2 and 3 ask for estimates of the percentage of features that are implemented not because they are specified as requirements by users, but because it is assumed that they might be needed – the higher the percentage the higher the (potential) overhead. Question 4 measures an estimate of the degree to which it is possible to determine the requirement behind any given piece of code. A high degree could indicate low overhead, because it would indicate the majority of the code is based on specified requirements. Note that the question asks about the ability to trace between requirements and code, not about the degree to which such tracing is actually performed. Therefore, it should not matter how much effort it would cost. Question 5 is a direct estimate of perceived unnecessary work.

3.1.2.5 Process Agility

```
1. Do you have team meetings to reflect on development practices and possible
improvements to them?
      ( ) Never
      ( ) Seldom
      ( ) Sometimes
      ( ) Often
      ( ) All the time

2. If you do you have team meetings to reflect on development practices and possible
improvements to them, how often do they typically take place?
      ( ) Rarely or never
      ( ) About once every... days:
```

```
3. Do you feel you are encouraged to suggest potential improvements to development
practices?
       ( ) Not at all
       ( ) Somewhat
       ( ) Moderately
       ( ) Strongly
       ( ) Very strongly

4. How often does the team make changes to its development practices that are
initiated...
                                       Never  Seldom Sometimes    Often  All the time
by the team itself?                    _____  _____  _____        _____  _____
by outside forces, e.g. management?    _____  _____  _____        _____  _____
```

Questions 1 and 2 measure the frequency of team meetings with the purpose of reviewing and improving development practices, while question 4 measures the frequency of actual changes to practices. Question 3 asks respondents about how comfortable they are with suggesting improvements to development practices. For all questions, a higher response indicates higher process agility.

3.1.3 Measuring Conceptual Modeling Practices

Measuring conceptual modeling practices is challenging because the concept of conceptual modeling is not understood the same way by all software development practitioners. In particular, "conceptual model" can be interpreted as "any model of a conceptual, i.e., abstract, nature". In this sense, models that are used to describe the design and implementation of software components could be considered to be conceptual models. However, the current research is interested specifically in models about "aspects of the physical and social world around us" (Mylopoulos 1992), i.e., in models about the domain, in contrast to models about the system under development.

In order to make sure that respondents consistently understand this distinction, the survey contains brief definitions of and examples for both conceptual models (models about the domain)

and design models (models about the system under development). Then, all questions about modeling are asked both for conceptual models and design models, highlighting the contrast throughout the survey. The disadvantage of this approach is that the survey gets significantly longer by the duplication of a number of questions. The advantages are that subjects are more likely to remain aware of the distinction between the two types of models throughout the survey, and that additional information about design modeling practices is gathered by the survey.

The survey contains questions about the following areas with respect to both conceptual and design modeling (the exact questions are shown in Appendix A – Survey Questions):

- Modeling methods (i.e., grammars) used: ER diagrams, class diagrams, use case diagrams etc.[9]
- The media on which models are stored: computer files, paper, white boards etc.
- The software packages that are used to perform modeling: word processors, spreadsheet software, graphics packages etc.
- What happens to models after they have been used: they are discarded, kept for future personal reference, kept for general team use etc.
- Whether there are specific rules or guidelines on how to do modeling: about which methods to use, when to create models, how to create models etc.

3.1.4 Measuring Domain Understanding Issues and Model Use

After measuring agility and conceptual modeling use, this part of the survey can be viewed as outcome-oriented. The key purpose of the questions in this section is to capture some information about how respondents use conceptual modeling in challenging situations, and whether they experience deficiencies in domain understanding. It should be pointed out that the current state of the research does not provide enough theoretical insights to allow for detailed

[9] Note that there are concepts or types of information about the domain that are not models in the strict sense, such as use cases, persona (Haikara 2007) or user stories. Thus, the survey does not specifically ask subjects about them.

examinations of causality between agility, conceptual modeling and domain understanding issues. For example, if the results were to indicate that high degrees of agility correlate with low conceptual modeling use and significant domain understanding issues, then it would be unclear whether the agile nature of the projects, the low conceptual modeling use, a combination of the two or some other unknown variables cause the domain understanding issues. Instead of explaining causality, this part of the survey merely attempts to provide some additional data related to the key variables of agility and conceptual modeling use.

Specifically, the survey contains questions about

- how often developers have to go back to users to get information that has previously already been provided by them
- the degree to which more conceptual models are used or more changes are made to conceptual models when the domain and/or requirements become increasingly complex
- how often conceptual models are used as communication tools with users, new team members, members of other teams and management
- the degree to which existing conceptual models are used to understand unfamiliar domains and requirements
- the frequency with which the developer has to provide the same information to other developers repeatedly

The exact questions are contained in Appendix A – Survey Questions.

3.1.5 Measuring Demographic Information

The final part of the survey asks respondents questions about demographic information with regard to their project, their organization and themselves. The key purpose of these questions is to measure potentially relevant control variables. In addition to this, one question measures the perception of agility (as a validation of the measure of agility described in section 3.1.2),

respondents can optionally provide contact information in case they are potentially willing to participate in follow-up research, and can provide comments or feedback about the survey itself.

The following items are measured:

Project-specific information:

- Duration of the project so far
- Additional duration for which the project is expected to last
- Number of developers and other team members involved in the project, both full-time and part-time
- Which (if any) specific development method is used, and how closely that method is followed (if applicable)
- How agile the respondent perceives the project to be

Organization-specific information:

- Number of employees
- Industry

Demographic information about the individual:

- Highest level of education
- Years of experience in software development and at the current position
- Most closely matching job description (business analyst, programmer/software engineer, team lead, project manager, quality assurance, product manager, other)
- Number of projects the respondent is currently involved in

The detailed questions are listed in Appendix A – Survey Questions.

3.2 Survey Execution

3.2.1 Target Population and Survey Platform

The population of interest for the survey is that of software development professionals. This primarily includes software developers. Other terms that are frequently used for the same job category include software engineers or programmers. In addition to developers, professionals filling other related roles in development projects are also in the target group. These roles include business analysts, team leads, project managers, quality assurance professionals (testers) and product managers. Professionals in these roles can typically be expected to be sufficiently knowledgeable about the development project to meaningfully respond to most of the survey questions. In case a respondent is not sufficiently involved in a specific area of the development process to answer some of the questions, the survey encourages the subject to skip these questions.

In order to reach this target group, the Web is a suitable delivery mechanism for the survey. Given the nature of their work, software development professionals can clearly be expected to be online. Therefore, providing the survey online should not create a selection bias of respondents. Furthermore, compared to alternatives such as the distribution of a paper-based survey in the mail, it should encourage participation in the survey and increase response rates due to the easy accessibility of the survey through hyperlinks. Instead of creating the technical infrastructure for the survey from scratch as was typical just a few years ago (Schleyer and Forrest 2000), one of several now existing survey sites, Survey Gizmo (http://www.surveygizmo.com/), was used. The choice of the specific site was driven by the extensive feature set supported, and by the availability of free unlimited surveys for research projects by (graduate) students.

As indicated above, responses by all developers, regardless of the level of agility in their projects, are desirable. This does not only allow the comparison between conceptual modeling practices between more agile and less agile development, but it also makes sure that there is sufficient variability in the degree of agility to test the measurement instrument of agility.

The whole survey is broken down into a number of web pages, with a progress bar informing respondents about their progress in the survey. The first page serves as the consent form and includes general information about the survey and contact information to reach the researchers. It requires respondents to indicate that they are at least 19 years of age, understand their rights as subjects in this research, and freely consent to participate. The following four pages contain the questions measuring agility, conceptual modeling use, issues with domain understanding and demographic information. The final page of the survey thanks subjects for their participation, asks them to consider forwarding the survey to other developers from their professional network, and contains a summary evaluation of their response (see section 3.2.3).

3.2.2 Pilot Tests

In order to test the technical platform of the survey and the understandability and relevance of the questions in the survey, two rounds of pilot tests of the survey were performed. The test subjects in the first round were colleague researchers in various related fields. The result of the first round was a technically stable and usable survey with improved (e.g., with regard to clarity) questions. In the second round, professional developers from the personal network of the author were asked to respond to the survey. Again, some survey questions were improved as a result of the feedback. At the end of this phase the survey was considered to be sufficiently clear and usable.

One of the main criticisms of the survey in both rounds of pilot tests was its length. As reported by the respondents and also measured by the survey tool, the typical time it took subjects to answer the over 50 questions of the survey was around 15-20 minutes. While this may seem perfectly acceptable for empirical research conducted with paid undergraduate students as subjects, it is a significant obstacle for the participation of busy software development professionals. Unfortunately, the objectives of the survey do not allow a smaller number of questions as is typically found in popular web-based polls and surveys. For example, multiple items per concept (e.g., the key dimension of agility) are necessary in a scientific survey in order to achieve validity and reliability of the measurements.

In order to encourage the participation in the survey despite its length, respondents are promised two main incentives in exchange for responding to the survey: They receive an (approximate) evaluation of the agility of their project, based on their answers, and a summary analysis of the answers of all respondents so far to selected survey questions.

3.2.3 Survey Promotion and Sampling Strategy

The survey was promoted using several different approaches. Initially, printed invitation letters explaining the purpose of the survey and containing a short link to the survey were distributed at the monthly meetings of two relevant local associations, one composed of developers in agile development, the other of software developers in general. The last page of the survey (which displays the computed agility score and a link to the summary results) encourages respondents to share a link to the survey with other developers in their professional network that might be interested in taking the survey (snowball sampling).

Furthermore, prominent bloggers and Twitter users in the area of software development were contacted by the author with a request to post a link to the survey. Unfortunately, this led to only very limited results, which can be explained by two main factors: A general "fatigue" with available online surveys, and the unusual length of the survey (when compared to non-scientific online surveys and polls). Finally, respondents of a previous survey about software development by a colleague researcher who had indicated their interest in future surveys were contacted via email, with a personally addressed initial invitation email, and a reminder email two weeks later.

This sampling strategy raises concerns about external validity, which "refers to the generalizability of sample results to the population of interest" (King 2005). In other words, are the results of the survey representative of the situation in real (agile) IS development? While the sampling strategy as a combination of convenience sampling and snowball sampling is not perfect and potentially raises concerns about the external validity of the results, it is the best possible approach in the absence of a general and accessible directory of development professionals. In order to allow for the analysis of potentially resulting response biases, the invitation links to the survey contained a different token for each of the different groups of subjects described above. Therefore, it is possible to examine whether there are any significant differences between the different groups. This turned out to not be the case. In addition to this, an examination of the demographic information provided by the respondents can be used to determine whether the sample of respondents seems obviously different from the general population of interest.

Finally, the survey tool provides information about all incomplete survey responses, including the page of the survey on which each subject abandoned the survey. Most subjects who chose to abandon the survey did so on the first page of the survey (the consent form) or on the second

page of the survey (questions to measure agility) (for detailed numbers, see section 3.3.1). This behaviour is an indication that the questions about model use, which come later in the survey, did not create a response bias, for instance by discouraging respondents who are not interested in modeling from completing the survey.

3.3 Results

3.3.1 Survey Respondents

The survey yielded 118 usable responses between February and September 2012. The survey link has been accessed approximately 400 additional times. The majority of these accesses were abandoned on the first page of the survey, i.e., the consent form (approximately 230 times), or before completion of the second page of the survey (approximately 150 times), which contains only questions to measure the degree of agility. Fewer than 20 respondents went on to pages 3 or 4 of the survey. In other words, the majority of respondents who chose to not complete the survey did so without seeing any of the questions of the survey, or at least without seeing any questions related to conceptual modeling. Therefore, there is no reason to suspect that the majority of abandoned survey responses introduces a bias to the responses. Such a bias might have been an issue if for instance developers interested in modeling complete the survey with a higher percentage than developers who are not interested in modeling.

An evaluation of the response rate of an online survey is difficult. In particular, there is no way to determine how many individuals had access to and read the invitations to the survey provided in paper form, via blog or twitter invitation and via email by the researcher or by other survey respondents. Of the approximately 520 individuals who accessed the URL of the survey, about 120 completed the survey, which results of a completion rate of about 25%. While 120 responses

might not sound very impressive, it is actually a relatively significant number of responses for the target group of software development professionals. For example, Dobing and Parsons (2006) ran a survey about the usage of the modeling language UML targeted to system analysts, and managed to achieve only 171 usable responses, even though the survey was promoted by the Object Modeling Group (OMG), an organization with approximately 800 member companies.

Most respondents are highly educated and have a significant amount of professional experience. Only about 10 percent of respondents have no college degree (see Table 1), and the average and median number of years of work experience are 13.8 and 11 years, respectively.

The majority of respondents described their main role or job function as programmer or software engineer (56%), team lead (31%), business analyst (23%), project manager (21%), quality assurance professional (11%) or product manager (8%). Respondents were asked to select all roles that applied, resulting in a total percentage of over 100% (see Table 2). There was a number of additional roles that one or few respondents wrote down, including software architect, consultant, and CTO.

Table 1: Education of Respondents

Highest level of education attained	Percentage
Graduate degree (PhD)	5%
Graduate degree (Master)	36%
Some graduate school	9%
College degree (Undergraduate)	40%
Some college	7%
High school diploma	2%
Some high school	1%

Table 2: Main Roles/Job Functions of Respondents

Main roles/jobs functions	Percentage *(note: multiple answers possible)*
Programmer/software engineer	56%
Team Lead	31%
Business Analyst	23%
Project Manager	21%
Quality Assurance	11%
Product Manager	8%

Respondents reported to be in their current jobs on average for 5.1 years (median: 4). At the time of the survey, 31% of respondents were only involved in one software development project, while 29% were involved in two projects (at any stage), 16% in three projects and 23% in more than three projects. Only 2 respondents (1.5%) were currently not involved in a software development project at all.

The organizations that respondents worked for come from a wide range of industries. Most notably, 24% are from the software industry, 11% from finance, banking or insurance, 7% from the computer industry and 7% from the Internet industry (see Table 3).

Table 3: Industries of the Respondent's Organizations

Industry	Percentage of respondents
Software	24%
Finance/banking/insurance	11%
Computers	7%
Internet	7%
Consulting	5%
Government/military	5%

Industry	Percentage of respondents
Healthcare/medical	5%
Telecommunications	4%
Education	3%
Entertainment/recreation	3%
Retail	3%
Various other industries	23%

The median company size was 600 employees, while the average company size was a surprisingly high 16,429. A closer look at the data, and at the answers to an explicitly optional questions that some of the respondents provided (Which company do you work for?), explains this discrepancy: While most respondents work for small or medium-sized companies, a significant number of respondents work for very large international corporations with hundreds of thousands of employees, such as IBM, Hewlett-Packard or Bank of America.

3.3.2 Data Analysis Procedure

3.3.2.1 Introduction

The survey tool Survey Gizmo exports all survey responses into a comma separated file (.csv). This file was edited in Microsoft Excel to normalize the raw data, and subsequently imported into the statistical software package PSPP (http://www.gnu.org/software/pspp/), an open source alternative to the commercial software SPSS, for actual statistical analysis.

Each of the five key components of agility was normalized into a range from 0 to 5, and the resulting total measure of agility calculated as the arithmetic mean of the five components. Within each component, each question measuring the component was normalized into a range from 0 to 5 and weighted equally with the other questions measuring the same component. For

some questions the normalization was not trivial. For example, consider the following question contributing to the measurement of the ability of the project to quickly respond to recognized changes:

```
4. In your current or most recent project, consider the backlog of all changes not
deemed to be critical that have been requested by users. Assuming no new changes, how
long would it approximately take to incorporate all change requests (if no other
development would occur at the same time)?
      days _________________
      weeks _________________
      months _________________
```

The raw response data contains (potentially empty) values for the number of days, weeks and/or months[10]. This data is first converted into a number of days (by multiplying the number of months by 30, the number of weeks by 7, and adding the results to the number of days. Then, the same conversion into days is done for the question about the duration of the project so far. In a third step, the duration of the project is divided by the duration required to work through backlog. Finally, the median and standard deviation of the resulting distribution of values are calculated, based on which the cut-off points for the assignment of a value between 0 and 5 are computed, and a value is assigned accordingly for each response value. Similar transformations are performed for the other variables.

3.3.2.2 Data Transformation

This section describes the process by which the raw response data was transformed into the 0-5 ranges in more detail. For the purposes of that transformation, there are three different types of questions (or more precisely, response formats) in the survey: Likert-scale responses, open

[10] In response to feedback from respondents in the pilot phase, the survey allows respondents to answer in different units of time (and in combinations of units). This makes it unnecessary for respondents to convert between units of time. The pilot has shown that some subjects (particularly of smaller projects) tend to think in days or weeks, while subjects in larger projects often think in terms of weeks or even months.

numerical responses, and multi-part responses. Of the 17 individual items that measure the five key dimensions of agility, 9 fall into the first category, 6 into the second, and 2 into the third.

Likert-scale responses are handled most easily. All Likert-scale questions in the survey (items ER_1 and ER_3 for Early Recognition, TANG_2 for Tangibility, LOWOH_1, LOWOH_2 and LOWOH_3 for Low Overhead and PA_1, PA_3 and PA_4 for Process Agility) have a 5-item response scale, which is translated directly into the values 0, 1.25, 2.5, 3.75 and 5.

Open numerical responses are more complex to transform. For example, subjects' responses to item ER_2 ("If you do provide users with early versions of the software, how often do you typically do so?") range between 1 and 720 days, with an average of 54, a median of 28 and a standard deviation of 98. Table 4 lists descriptive statistics of the raw response data for the six items with open numerical responses.

Table 4: Descriptive Statistics of Raw Response Data for Open Numerical Questions

	ER_2	ER_4	QR_1	QR_2	QR_3	PA_2
Minimum	1	1	0.5	1	0	1
Maximum	720	720	180	150	1029	180
Average	54	57	14	15	27	36
Median	28	28	3	6	6	14
Standard Deviation	98	103	27	29	103	50

It is not clear how these numbers should be transformed into ranges from 0 to 5. Some experimentation led to the boundaries described in Table 5, which resulted in the distributions for the resulting transformed data with the descriptive statistics shown in Table 6.

Table 5: Transformation of Raw Data into 0-5 Ranges

Raw Response Data Range	Assigned Value
0 to 2/5 * Median	0
2/5 * Median to 4/5 * Median	1
4/5 * Median to 6/5 * Median	2
6/5 * Median to 6/5 * Median + Standard Deviation	3
6/5 * Median + Standard Deviation to 6/5 * Median + 2* Standard Deviation	4
> 6/5 * Median + 2* Standard Deviation	5

Table 6: Descriptive Statistics of Transformed Data for Open Numerical Questions

	ER_2	ER_4	QR_1	QR_2	QR_3	PA_2
Average	2.97	1.80	3.02	3.29	2.05	1.86
Standard Deviation	1.68	1.92	1.74	1.49	1.87	1.60

The only exception is item QR_3, for which this transformation led to an unsatisfactory resulting distribution due to the high standard deviation of the raw data combined with a very low median. Note that the "raw data" shown for QR_3 is already a computed value, because it is the ratio of the duration of the project and the estimated duration it would take to work through the backlog of non-critical changes (see section 3.3.2.1). For QR_3, the last two range boundaries where changed to 6/5 * Median + 1/10 *Standard Deviation and 6/5 * Median + 1/5 *Standard Deviation.

This adjustment, as well as the boundaries described in Table 5, are somewhat arbitrarily chosen. To ensure that these decisions did not unduly influence the statistical results, a sensitivity analysis was performed, in which the range boundary values were manually adjusted slightly.

The statistical results did not change in a meaningful way. This should not be particularly surprising, because for most of the five dimensions of agility, questions with open numerical responses are paired with Likert-scale response questions for increased validity. For example, for Early Recognition, item ER_1 ("Do you provide early versions...") has a Likert-Scale response, while ER_2 ("How often do you provide early versions...") asks for an open numerical response.

Items ER_5 and TANG_1 fall into the final category of response types, multi-part responses. For these types of questions, each possible response was assigned a numerical value, and the numerical values of all selected answers were added up to the resulting agility score. For example, for ER_5 ("How do end users give you feedback on the software or request changes to it? Check all that apply."), the response "Users approach developers" was assigned a value of 3, while "Regular meetings" was assigned a value of 2, since the former was deemed to reflect a larger contribution to the early recognition of the need for changes than the latter. If the sum of all selected responses exceeded 5, the value 5 was assigned to the resulting variable.

3.3.3 Measuring Agility

As discussed in section 3.1.1, technical validation of the measurement items for agility as typically performed in the IS literature for measures of attitudes or intentions is not sensible, because the individual questions measuring the key components of agility measure different (even though related) facts and therefore cannot be expected to correlate perfectly. Consequently, the frequently practiced approach of removing individual measurement items that load lowest on the variable as a whole was not used. Despite this caveat, most measurement items correlate highly with their construct, as the following tables show.

Table 7: Correlations between Early Recognition of the Need for Changes (ER) and its Items (ER_1 to ER_5)

		ER_1	ER_2	ER_3	ER_4	ER_5	ER
ER_1	Pearson Correlation	1.00	.67	.54	.50	.27	.78
	Sig. (2-tailed)		.00	.00	.00	.00	.00
	N	118	118	118	118	118	118
ER_2	Pearson Correlation	.67	1.00	.46	.61	.35	.82
	Sig. (2-tailed)	.00		.00	.00	.00	.00
	N	118	118	118	118	118	118
ER_3	Pearson Correlation	.54	.46	1.00	.83	.07	.79
	Sig. (2-tailed)	.00	.00		.00	.44	.00
	N	118	118	118	118	118	118
ER_4	Pearson Correlation	.50	.61	.83	1.00	.16	.85
	Sig. (2-tailed)	.00	.00	.00		.08	.00
	N	118	118	118	118	118	118
ER_5	Pearson Correlation	.27	.35	.07	.16	1.00	.49
	Sig. (2-tailed)	.00	.00	.44	.08		.00
	N	118	118	118	118	118	118
ER	Pearson Correlation	.78	.82	.79	.85	.49	1.00
	Sig. (2-tailed)	.00	.00	.00	.00	.00	
	N	118	118	118	118	118	118

Table 8: Correlations between Tangibility (TANG) and its Items (TANG_1 and TANG_2)

		TANG_1	TANG_2	TANG
TANG_1	Pearson Correlation	1.00	-.07	.68
	Sig. (2-tailed)		.46	.00
	N	118	118	118
TANG_2	Pearson Correlation	-.07	1.00	.69
	Sig. (2-tailed)	.46		.00
	N	118	118	118
TANG	Pearson Correlation	.68	.69	1.00
	Sig. (2-tailed)	.00	.00	
	N	118	118	118

Table 9: Correlations between Quick Response to Recognized Changes (QR) and its Items (QR_1 to QR_3)

		QR_1	QR_2	QR_3	QR
QR_1	Pearson Correlation	1.00	.60	.07	.76
	Sig. (2-tailed)		.00	.43	.00
	N	118	118	118	118
QR_2	Pearson Correlation	.60	1.00	.16	.78
	Sig. (2-tailed)	.00		.08	.00
	N	118	118	118	118

		QR_1	QR_2	QR_3	QR
QR_3	Pearson Correlation	.07	.16	1.00	.62
	Sig. (2-tailed)	.43	.08		.00
	N	118	118	118	118
QR	Pearson Correlation	.76	.78	.62	1.00
	Sig. (2-tailed)	.00	.00	.00	
	N	118	118	118	118

Table 10: Correlations between Low Overhead / Leanness (LOWOH) and its Items (LOWOH_1 to LOWOH_3)

		LOWOH_1	LOWOH_2	LOWOH_3	LOWOH
LOWOH_1	Pearson Correlation	1.00	-.06	-.22	.39
	Sig. (2-tailed)		.51	.02	.00
	N	118	118	118	118
LOWOH_2	Pearson Correlation	-.06	1.00	.05	.67
	Sig. (2-tailed)	.51		.58	.00
	N	118	118	118	118
LOWOH_3	Pearson Correlation	-.22	.05	1.00	.53
	Sig. (2-tailed)	.02	.58		.00
	N	118	118	118	118
LOWOH	Pearson Correlation	.39	.67	.53	1.00
	Sig. (2-tailed)	.00	.00	.00	
	N	118	118	118	118

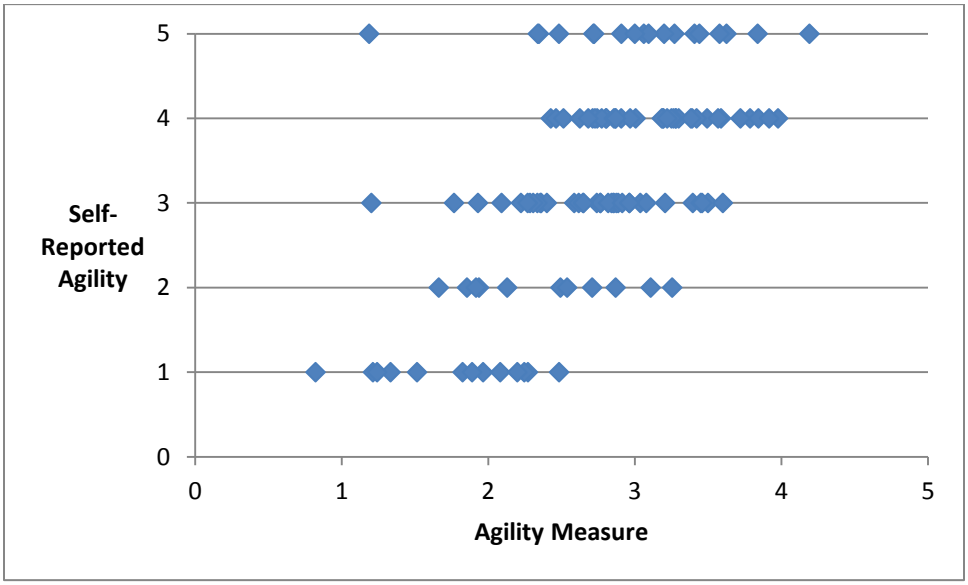
Table 11: Correlations between Process Agility (PA) and its Items (PA_1 to PA_4)

		PA_1	PA_2	PA_3	PA_4	PA
PA_1	Pearson Correlation	1.00	.74	.46	.41	.85
	Sig. (2-tailed)		.00	.00	.00	.00
	N	118	118	118	118	118
PA_2	Pearson Correlation	.74	1.00	.41	.34	.82
	Sig. (2-tailed)	.00		.00	.00	.00
	N	118	118	118	118	118
PA_3	Pearson Correlation	.46	.41	1.00	.47	.75
	Sig. (2-tailed)	.00	.00		.00	.00
	N	118	118	118	118	118
PA_4	Pearson Correlation	.41	.34	.47	1.00	.68
	Sig. (2-tailed)	.00	.00	.00		.00
	N	118	118	118	118	118
PA	Pearson Correlation	.85	.82	.75	.68	1.00
	Sig. (2-tailed)	.00	.00	.00	.00	
	N	118	118	118	118	118

To evaluate the validity of the measurement instrument of agility, an independent assessment of the degree of agility of the project of each respondent would be ideal. Since this is obviously infeasible, the best available alternative – the self-reported degree of agility – will have to suffice as an approximation of actual agility and as a standard of comparison for the proposed measurement instrument.

Figure 1 shows a scatter plot of the relationship between measured and self-reported agility.

Figure 1: Measured vs. Self-Reported Agility



The scatter plot shows a clear, yet imperfect correlation between the two variables. This is the expected outcome for a successful validation of the instrument. A very low or non-existent correlation would indicate that the instrument poorly reflects what practitioners perceive as agility, while a perfect correlation would render the whole exercise of administering the new measurement instrument unnecessary – if an instrument of approximately 20 questions gives exactly the same result as a single question (self-reported agility), then the latter would clearly be preferable.

A statistical analysis reveals a Pearson correlation of 0.59 with a two-tailed significance of 0.01. An analysis of variance (ANOVA) between the two variables results in an F-value of 19.16 with a significance of 0.01. Table 12 shows how the five measured key components of agility correlate with self-reported agility:

Table 12: Correlations between the Five Components of Measured Agility (AGILE_M) and Self-Reported Agility (AGILE_S)

	ER	QR	TANG	LOWOH	PA	AGILE_M	AGILE_S
Early Recognition (ER)	1						
Quick Response (QR)	0.38 (0.00)	1					
Tangibility (TANG)	0.26 (0.00)	-0.01 (0.93)	1				
Low Overhead (LOWOH)	0.22 (0.02)	0.13 (0.15)	0.11 (0.23)	1			
Process Agility (PA)	0.34 (0.00)	0.18 (0.05)	0.27 (0.00)	0.26 (0.00)	1		
Measured Agility (AGILE_M)	0.75 (0.00)	0.59 (0.00)	0.51 (0.00)	0.50 (0.00)	0.67 (0.00)	1	
Self-reported Agility (AGILE_S)	0.46 (0.00)	0.42 (0.00)	0.19 (0.04)	0.15 (0.11)	0.48 (0.00)	0.59 (0.00)	1

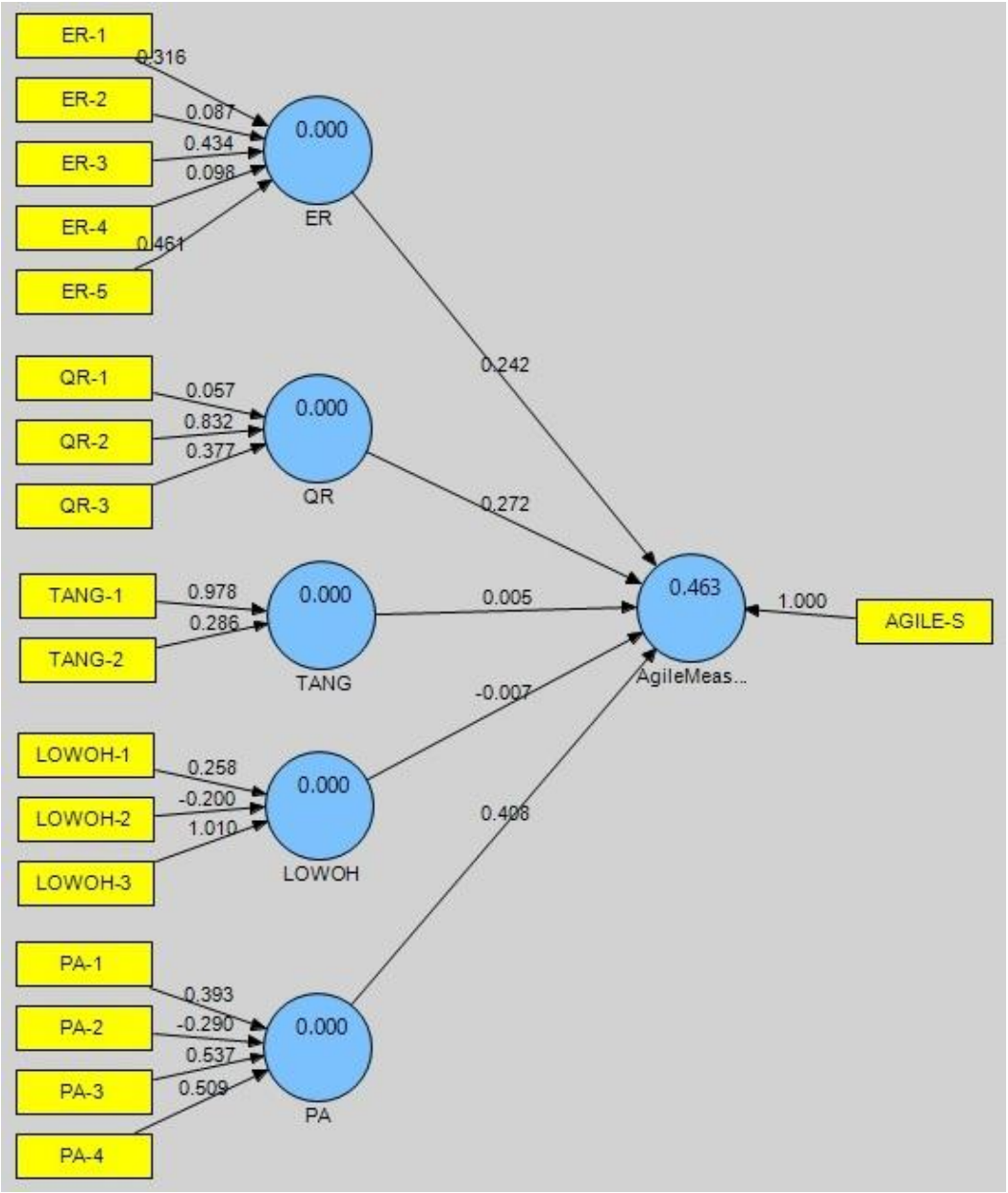
An analysis of variance (ANOVA) between the same variables shows F-values over 8 at a significance of 0.01 for all variables except tangibility and low overhead/leanness, which show lower F-values and significance (see Appendix B – Measured Agility vs. Self-Reported Agility).

For a more comprehensive analysis of the properties of the measurements and the relationships between the different variables (measured agility and its five dimensions, and self-reported agility), the partial least squares method (PLS) was used. As Vinzi, Trinchera et al. (2010) point out, PLS is useful to validate the measurement model and to estimate path coefficients in the structural model. PLS does not assume a specific distribution of the observed variables, does not

require large sample sizes, and is well suited to model formative constructs (Chin 2010), which makes it particularly appropriate for this study.

Figure 2 shows the results created by the PLS tool SmartPLS (Ringle, Wende et al. 2005). The model is fairly straightforward: "AgileMeasured" is a second-order latent variable with five formative constructs – the five dimensions of agility. This measured degree of agility is then approximated by self-reported agility ("Agile-S"). The five key dimensions of agility are also modeled with underlying formative indicators (the individual measurement items). As discussed in 3.1.2, even though the individual measurement items are conceptually in a more reflective relationship with the different key dimensions than the five key dimensions are with the concept of agility, the fact that they measure clearly distinct facts makes a reflective model unreasonable. To recapitulate the argument, for instance the construct of "Early Recognition" is measured by whether and how often early versions of the system are released to end users for either testing purposes or for productive use. While a frequent use of early releases for either of the two purposes could be argued to reflect the ability to recognize the need for changes early, it is perfectly reasonable to use early releases only for testing purposes. If "Early Recognition" was modeled as a reflective construct, then any survey responses indicating the use of early releases for only one purpose would make the measurement model appear unreliable, which clearly would not be the case.

Figure 2: PLS Results (Please refer to page 143 for a definition of the variables)



Most indicators contribute positively to their respective constructs. The exceptions are the items "LOWOH-2" and "PA-2". The former is the answer to the question "What percentage of non-functional features that you implement is not directly derived from user requirements, but added because they are likely going to be needed anyways?". The theoretical expectation is that a high percentage of features that are not directly derived from user requirements indicates significant

overhead, since estimates by developers about what users might eventually need are – due to the difficulty of predicting user needs – unreliable, resulting in the implementation of unnecessary features, and consequently overhead. While this reasoning seems to apply to "LOWOH-1", which is the same question about *functional* requirements, and results in a positive contribution of the indicator to the construct of "Low Overhead", the nature of *non-functional* requirements might explain why the contribution of "LOWOH-2" is negative. Jacobson, Booch et al. (1999) define non-functional requirements as "system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability". Due to the more technical nature of non-functional requirements, end users will have bigger difficulties recognizing and expressing them than they do with functional requirements. As Glinz (2007) points out, there are no standard methods in software engineering for the elicitation, documentation and validation of non-functional requirements. Therefore, it might actually reduce overhead and increase agility if developers incorporate non-functional requirements that have not been specified by users, because failure to do so might lead to extensive re-work later.

"PA-2", the second indicator that contributes negatively to its latent variable, measures the response to the question "If you do you have team meetings to reflect on development practices and possible improvements to them, how often do they typically take place?". The theoretical expectation behind the question is that more frequent meetings indicate higher process agility, because they give more opportunities to adjust the development process. However, an alternative explanation for the opposite (negative contribution) exists: Very frequent meetings might indicate a very heavy-weight (and thus non-agile) process, in which any (even minor) change to the development process needs to be decided on in an "official" meeting. In other words, fairly

infrequent reflection meetings might be a result of high autonomy of development team members, who are authorized to constantly improve their development process on a local level, and therefore exhibit a high degree of process agility.

The relationships between the five key dimensions of agility and agility as a whole as indicated by the path coefficients in the structural PLS model agree with the correlation analysis: "Process Agility", "Early Recognition" and "Quick Response" contribute most strongly with self-reported agility, while "Tangibility" and "Low Overhead" do so much less (the latter even with a very small negative contribution in the PLS model). Overall, the model finds an R-square of 0.463 for the explained portion of the total variation of self-reported agility.

3.3.4 Conceptual Modeling and Domain Understanding

To analyze the relationship between the degree of agility and conceptual modeling and domain understanding, a number of different approaches are possible. The general idea is to group respondents by their degree of agility, and to compare the survey responses between the different groups. Both self-reported and measured agility can be used to group respondents. The latter was chosen for the following analysis, because it more closely reflects agility as understood in this work, and because it allows a more fine-grained distinction between different degrees of agility due to its nature as a continuous numerical variable, in contrast to self-reported agility, which has only five possible values (1, 2, 3, 4 or 5).

A second decision to make is the number of agility levels that the respondents are grouped into. For example, respondents can be assigned to either "High" or "Low" agility, or to "High", "Medium" and "Low" agility. Due to the relatively small sample size of about 120 respondents, only two categories were chosen.

To ensure that these decisions do not unduly influence the results, a sensitivity analysis was performed. The results did not change significantly when self-reported agility was used instead of measured agility, or when the number of categories was changed (e.g., high/medium/low agility). This should not be surprising, given the relatively strong correlation between measured and self-reported agility (see section 3.3.3). Also, increasing the number of groups might increase the mean differences between the groups. However, it also decreases the sample size per group. Both effects should thus largely leave the statistical power of any analysis constant.

Since the survey contained questions about approximately twenty different aspects related to conceptual modeling and domain understanding, running multiple t-tests to find whether any mean differences between the "High" and "Low" groups are statistically significant would carry the increased risk of a type 1 error. Therefore, an analysis of variance (ANOVA) was performed instead.

Of all the variables measured (see Appendix C – Survey Results about Modeling/Domain Understanding), five resulted in mean differences between the "High Agility" and "Low Agility" groups that are statistically significant or close to being statistically significant, with p-values ranging from 0.00 to 0.09. Table 13 shows descriptive statistics for these five variables, and Table 14 contains the results of the ANOVA analysis.

The results for CM_GrCount (which measures the number of different conceptual modeling grammars used) show that more agile developers use a wider range of conceptual modeling grammars than less agile developers. Given the general tendency of the agile paradigm against extensive documentation and overhead (Beck, Beedle et al. 2001), this is a surprising outcome. With a p-value of 0.09 the difference between the HIGH and LOW groups is not quite

statistically significant with the typically chosen cut-off value of p=0.05. Therefore, the mean difference for this variable might be caused by chance, or the sample size might simply be too small to detect the difference with a lower p-value. Future research will need to give a conclusive answer to this question.

Table 13: Descriptive Statistics for Significant or Almost Significant Variables

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean: Lower Bound	95% Confidence Interval for Mean: Upper Bound	Minimum	Maximum
CM_GrCount	HIGH	59	1.92	1.29	.17	1.58	2.25	0	5
	LOW	59	1.53	1.21	.16	1.21	1.84	0	5
	Total	118	1.72	1.26	.12	1.49	1.95	0	5
RE_CLARIFY	HIGH	59	2.61	.97	.13	2.36	2.86	0	5
	LOW	59	3.07	1.27	.17	2.74	3.40	0	5
	Total	118	2.84	1.15	.11	2.63	3.05	0	5
CHANGE_TYPES	HIGH	59	1.71	1.18	.15	1.41	2.02	0	3
	LOW	59	1.32	1.15	.15	1.02	1.62	0	3
	Total	118	1.52	1.17	.11	1.30	1.73	0	3
COM_MA	HIGH	59	2.47	1.34	.17	2.12	2.82	0	5
	LOW	59	2.00	1.23	.16	1.68	2.32	0	5
	Total	118	2.24	1.31	.12	2.00	2.48	0	5
RE_EXPL	HIGH	59	1.93	.83	.11	1.72	2.15	0	4
	LOW	59	2.63	1.16	.15	2.33	2.93	1	5
	Total	118	2.28	1.06	.10	2.09	2.47	0	5

(Please refer to page 143 for a definition of the variables)

Table 14: ANOVA Results for Significant or Almost Significant Variables

		Sum of Squares	df	Mean Square	F	Significance
CM_GrCount	Between Groups	4.48	1	4.48	2.87	.09
	Within Groups	181.29	116	1.56		
	Total	185.77	117			
RE_CLARIFY	Between Groups	6.18	1	6.18	4.85	.03
	Within Groups	147.76	116	1.27		
	Total	153.94	117			
CHANGE_TYPES	Between Groups	4.48	1	4.48	3.31	.07
	Within Groups	156.98	116	1.35		
	Total	161.47	117			

```
#============================#==============#===#===========#=====#============#
#                            #Sum of Squares| df|Mean Square|  F  |Significance#
#============#==============#==============#===#===========#=====#============#
#COM_MA      |Between Groups#          6.64|  1|       6.64| 4.00|          .05#
#            |Within Groups #        192.71|116|       1.66|     |             #
#            |Total         #        199.36|117|           |     |             #
#------------+--------------#--------------+---+-----------+-----+------------#
#RE_EXPL     |Between Groups#         14.25|  1|      14.25|14.06|          .00#
#            |Within Groups #        117.53|116|       1.01|     |             #
#            |Total         #        131.77|117|           |     |             #
#------------+--------------#--------------+---+-----------+-----+------------#
```

Table 15 shows how many respondents in the "High" and "Low" groups reported the use of which conceptual modeling grammars.

Table 15: Conceptual Modeling Grammars Used

Use of conceptual modeling grammars (by 59 more agile and 59 less agile respondents)	**More agile (High)**		**Less agile (Low)**	
	Number	**Percentage**	**Number**	**Percentage**
ER diagrams	19	32%	15	25%
Class diagrams	19	32%	13	22%
Use case diagrams	24	41%	21	36%
UML activity/sequence diagrams	15	25%	11	19%
Process diagrams (e.g., BPMN/EPC)	17	29%	18	31%
Other	19	32%	12	20%
No modeling grammar used	6	10%	11	19%

All types of grammars except process diagrams were used by a higher percentage of the more agile respondents than the less agile respondents. One potential reason for this exception might be that process diagrams tend to be more complex than other types of diagrams, making them less practical in an agile context. Similarly, a smaller percentage of the more agile developers reported the use of no conceptual modeling grammars than the less agile ones. However, these

differences have to be interpreted carefully, since they are not statistically significant, and could thus be caused by chance. They are still interesting results, if only since the opposite outcome – significantly lower conceptual model use in agile development – did also not occur.

The variable "CHANGE_TYPES" measures how many different kinds of changes to conceptual models respondents make frequently when the domain or requirements become more complex. Out of four different kinds of changes (additions of new constructs, deletions of constructs, changes to existing constructs, and other), respondents in the "High" group on average reported 1.71 types, while respondents in the "Low" group on average reported only 1.32 types. Similarly to the variable CM_GrCount, the mean difference just missed the usual cut-off for significance of 0.05 at 0.07. Therefore, the same caution in the interpretation should be exercised.

A third variable that points to higher conceptual modeling use in agile development is "COM_MA", which measures on a 5-point Likert scale from 1 (almost never) to 5 (almost always) how often conceptual models are used as a communication tool to explain parts of the domain or requirements to management. The more agile respondents averaged a value of 2.47, compared to a value of 2.00 for the less agile ones.

While the three variables discussed so far provide some evidence that agile developers use conceptual modeling practices more extensively than less agile developers, the remaining two variables with statistically significant mean differences concern the outcome of development practices on domain understanding and communication about the domain.

The variable "RE_CLARIFY" measures on a 5-point Likert scale from 1 (almost never) to 5 (almost always) how often respondents "have to go back to users (or whoever else provides requirements to you, e.g. a product manager) for clarifications about requirements, even though

the needed information was already provided previously, e.g. to another developer". The more agile respondents averaged a value of 2.61, while the less agile ones averaged 3.07. Similarly, the variable "RE_EXPL" measures on the same scale the response to the question "How often do you say something like 'I have already explained this to you / another developer/ in a meeting'?" Again, respondents in the "High" group averaged a lower value (1.93) than respondents in the "Low" group (2.63).

The remaining variables show mean differences that are not statistically significant with at least a 90% probability (Appendix C – Survey Results about Modeling/Domain Understanding contains all results). This obviously does not conclusively imply that the level of agility makes no difference for these variables, but it could simply mean that the sample size was insufficient to detect small effect sizes given the level of variance in the responses.

Obviously, the survey cannot determine the causal reason behind these differences. For example, it could be that the use of agile methods reduces the number of communication and domain understanding problems, for example because of more efficient and effective communication. However, alternative explanations cannot be ruled out. For example, other (potentially not measured) differences could result in both the choice of a more agile development approach and the outcome of fewer communication and domain understanding problems. One such obvious variable is project size: Smaller project size might lead to a preference for agile development approaches (Boehm 2002), and at the same time cause a lower occurrence of domain understanding and communication problems due to the lower complexity of such smaller projects. However, the reported project size between the groups "High" and "Low" is not significantly different – while the more agile respondents reported a somewhat smaller project

size (on average 26 team members, compared to 33 for the less agile respondents), the duration is of the project was somewhat higher for the more agile respondents (32 months vs. 28 months).

3.3.5 Summary and Implications

In summary, the survey provided evidence that the conceptualization of agility as a formative construct of

- early recognition of the need for changes
- quick response to recognized required changes
- process agility

and to a lesser degree of

- high degree of tangibility of intermediate results
- low overhead/leanness

reflects subjectively perceived agility well, and can be used as a basis for a more objective and generally applicable measurement instrument than existing measurement scales of agility (e.g., Maruping, Venkatesh et al. (2009)).

Furthermore, the survey supported the hypothesis that practitioners using agile development methods do in fact use a wide range of conceptual modeling practices, even though the agile paradigm discourages most forms of documentation as potentially unnecessary overhead (Beck, Beedle et al. 2001), and even though most agile development methods do not prescribe conceptual modeling use. Specifically, the results indicated that more agile developers tend to use a higher number of different modeling grammars, use them more intensively by making

more changes to their conceptual models when faced with increasing complexity[11], and use conceptual models more when communicating about the domain or requirements to other stakeholders (especially management).

This higher level of conceptual model use coincides with (even though it is not necessarily causally related to) an improved level of domain understanding and more effective communication about the domain and requirements for the more agile respondents, as the less frequent need to go back to users for clarifications on requirements and the less frequent need to explain the same thing repeatedly indicate.

Based on these results, the next logical step in this research is to investigate more closely *how* the use of conceptual modeling practices can improve domain understanding and communication about the domain in an agile setting.

[11] Note that, as discussed above, these two differences are not quite statistically significant at a p-value of 0.05, but only at 0.07 and 0.09, and are thus somewhat more likely to be caused by chance.

Chapter 4: The Impact of Conceptual Modeling on Agile Practices

4.1 Introduction

4.1.1 Goal of the Experiment

The survey described in the previous chapter provided evidence that conceptual modeling practices are in fact used in agile development. Based on this, a logical next step is to investigate more closely how conceptual modeling might influence agile development. For this purpose, an experiment that simulated agile development was performed, with the goal to study the effects of the treatment (the use of conceptual modeling) in-depth.

In the IS literature, a significant amount of empirical work on conceptual modeling has been performed. Wand and Weber (2002) provide a framework to classify research on conceptual modeling. They describe four different categories of research – grammars, methods, scripts and context, list examples of existing work and describe opportunities for future research in each of these categories.

Very little of this wide range of research is concerned with the use of conceptual models in groups or teams. In Wand and Weber (2002), only "social agenda factors", a single subcategory of one of the four categories of research, explicitly deals with questions that relate to the use of conceptual models in a social, i.e., group, setting. Also, most of the research in this area reviewed by the paper concentrates on different philosophical perspectives behind the use of conceptual models (such as positivist vs. interpretivist points of view), rather than on phenomena of group interaction. Similarly, in the framework for empirical evaluation of conceptual modeling techniques in Gemino and Wand (2004), and in the literature reviewed in that study, the direct study of the use of conceptual models in group settings plays a very limited role.

However, in the context of the research described here – agile IS development – the setting of conceptual model use in teams or groups is of critical importance. Two of the four elements that the agile manifesto (Beck, Beedle et al. 2001) values highly – "Individuals and Interactions" and "Customer Collaboration" – directly involve communication between different individuals. This focus on communication can be interpreted as a consequence of the attempt to reduce unnecessary overhead by reducing documentation. In the absence of formal documents, knowledge transfer in agile development happens mostly through direct communication between developers and users (Rubin and Rubin (2010), Abrahamsson, Warsta et al. (2003)).

The lack of significant research on conceptual modeling use in group or team settings makes this research by necessity exploratory in nature. Andersen, Richardson et al. (1997) discuss model building in groups and point out that it is not even clear which theoretical framework to apply – they for instance suggest various different factors from widely different theoretical backgrounds that could predict the success of model building, from systems thinking to group communication climate and group structure. An example of such exploratory research is Easterbrook, Yu et al. (2005). That paper describes a study into whether an explicit representation of different viewpoints held by different stakeholders in conceptual models leads to models of a higher quality. In the study, two teams of subjects were asked to create a model of the same domain, the treatment group using viewpoints, while the control group was asked to create a unified model. The authors collected purely qualitative data from the modeling process in both teams to analyze the advantages (e.g., improved understanding of the problem domain) and disadvantages (e.g., increased overhead) of using viewpoints.

The experiment described here takes a similar approach. Rather than attempting to achieve large sample sizes that would allow for sufficient statistical power to analyze quantitative differences

between treatment and control groups, the focus is on qualitative data collected in a more realistic experimental setting. There are two main reasons for this: Firstly, the immature state of research into conceptual modeling use in teams makes concrete and testable hypotheses difficult to come up with – more can be gained from in-depth analysis of qualitative data. The insights gained can then be used as a basis for hypotheses for future research. Secondly, the complexity of the subject of research – the use of conceptual models in a group setting in agile IS development – necessitates a rather complex experimental design to achieve any level of realism. This includes groups of subjects working together on non-trivial tasks in an agile context, ideally in multiple sessions, to allow for familiarization with the task and domain and to capture longitudinal effects. Achieving large sample sizes for statistical power in such a complex experiment would be very challenging (not just financially, but also from an administrative and subject recruitment perspective).

In summary, the goal of the experiment is to create a scenario that is a realistic (albeit simplified) representation of agile development, and to study the effects of the treatment (the use of conceptual modeling) in-depth by analyzing qualitative data gathered from observing subjects perform the experimental task. As Tichy (2000) points out, empirical work of this nature cannot be expected to be perfect and to give the "final answer" about a research question, but instead can contribute to knowledge by "small steps in the right direction".

4.1.2 Design Considerations

Based on the framework for empirical conceptual modeling research in Gemino and Wand (2004), for the design of an experiment various decisions regarding the independent variables of the study and the outcome dimensions of interest have to be made. Since the goal of the experiment is to examine the difference between treatment (conceptual model use during the

task) and control (no conceptual model use), several factors are simply held constant between both conditions. These factors are discussed in the next section. Specifically, this includes the question of the content used in the study (e.g., which types of case materials and models are used), which grammar constructs and rules are used, and on which media the content is delivered.

One of the key decisions is the nature of the tasks, which can either be model creation ("writing") or model use/interpretation ("reading"). While both alternatives (or potentially a combination of both) is worth studying in this context, the decision was made to focus on model use/interpretation. The reason for this decision is that the key aspect of interest of the experiment, the use of conceptual modeling practices in teams in the context of agile development, can be most easily studied in model interpretation/use, because this more naturally involves multiple stakeholders than model creation, which frequently is performed by a specific role (e.g., system analyst) *after* communication with other stakeholders takes place.

Besides the nature of the task, the type of subjects is another key consideration. Gemino and Wand (2004) classify subjects as either novices or experts in both modeling expertise and domain expertise. In the context of this study, the decision was made to use student subjects (and thus subjects that are closer to being novices than experts in both dimensions). While this decision is obviously partially made out of reasons of practicality (the relative ease of recruiting student subjects, compared to the challenge and cost of recruiting professionals involved in IS development), it can be argued that this decision is consistent with the goals of the study, which is one of the key decision criteria suggested by Compeau, Marcolin et al. (2012) regarding the use of student subjects in IS research. Since this study is exploratory in nature, and the goal is to gather first evidence on how conceptual model use can impact agile development practices, the

usefulness of the study does not critically depend on complete similarity between the subjects and the population of interest (professionals involved in IS development). Moreover, a very high level of subject's expertise in the domain is not desirable in a model interpretation/use experiment: If the subjects are experts in the domain, they are more likely to not use the provided conceptual models extensively, and instead to rely on their own knowledge of the domain, which in effect would render the treatment too weak. Finally, the model interpretation/use nature of the experiment also makes very high levels of modeling expertise unnecessary, since subjects do not have to create, but only use models. In the context of agile development, in which communication between all stakeholders is considered critical (Beck, Beedle et al. 2001), models should be simple and understandable enough for users with minimal technical and modeling skills. Notwithstanding these considerations, in subject recruitment care is taken that subjects have taken courses that teach the knowledge and skills required for the task.

Regarding the outcome dimensions described by Gemino and Wand (2004), decisions about the focus of observation and about the criteria for comparison have to be made. The focus of observation can either lie on the product of the task (i.e., the outcome or deliverables), or on the process of the task performance (i.e., how subjects arrived at the outcome). For this study both types of observations are used, while a special focus is on the process of the task performance, because it is expected that this might yield deeper insights into the impact of conceptual modeling than the static outcome of the task alone. As Gemino and Wand (2004) point out, the criteria for comparison depend on the specifics of the focus of observation. They are therefore discussed in detail below. They suggest effectiveness and efficiency measures of the conceptual modeling use (i.e., how well does the use of conceptual modeling techniques support the subjects

in the performance of their task, and how many resources (specifically, time) do the conceptual modeling techniques require). Both are part of the criteria used in the experiment.

4.2 Experimental Design

4.2.1 Initial Design

In agile development, a project is typically broken down into many short, fixed-length iterations. An iteration might have a duration of 1-2 weeks (Beck 1999). At the end of each iteration, a new version of the system under development with a small number of additional features is available. While (future) users of the system are encouraged to examine the system at the end of each iteration, it is typically not formally released as an actually usable system (with partial functionality) at the end of each iteration. Instead, such more formal releases occur after a typically fixed number of iterations (Ceravolo, Damiani et al. 2003). In the project simulated in this experiment, the iteration length is one week, and the system is released every four iterations (i.e., every four weeks).

The experiment simulates the first two release planning meetings in an agile IS development project. In each of the two meetings, subjects were asked to plan the upcoming release and its first iteration. Agile development methods typically use the concept of user stories to specify requirements (Savolain, Kuusela et al. 2010). A user story describes a single requirement from the perspective of a user in 1-2 brief sentences. User stories should be concise enough to fit on index cards. In each meeting, subjects began with a list of user stories that were to be implemented, and received additional input from developers, users and the project sponsor (i.e., typically the executive in charge of the budget funding the project). Subjects were asked to prioritize the user stories by selecting which user stories should be implemented in the upcoming

release and its first iteration. In addition, subjects were asked to identify any existing ambiguities, contradictions and duplications in the user stories provided.

The domain of the system under development was carefully chosen to be neither too familiar nor completely unfamiliar to subjects in the study. Khatri, Vessey et al. (2006) compare the influence of domain familiarity on the performance of different tasks, by presenting subject with either a completely unfamiliar domain (hydrology concepts as they are used in the oil industry) or with a very familiar domain (concepts from a sales domain such as product, order and sale). They find that the familiarity of the domain makes a strong difference in problem solving tasks. Therefore, for this experiment a domain and system was chosen that should be fairly familiar to subjects (business school students) – a time reporting and billing system as it might be used by a small consulting firm. Since the details of the system are made up for this study (and to a high degree arbitrarily chosen), none of the subjects could be familiar with the precise requirements. Appendix D –Experimental Materials contains the case description and the model of the domain that was provided to the treatment groups.

The subjects in the experiment were briefed in their instructions to play three different roles, representing both the technical and the business side:

1. The development team lead, representing the technical side as the representative of the developers
2. The product owner, representing the future users of the system (part of the business side)
3. The project manager, who is in charge of the project overall and of the simulated meeting.

Subjects were asked to come to an agreement about their decisions and to document their reasoning behind them for the project sponsor. A group size of three subjects per group was chosen so that there would be sufficiently complex communication between the group members

– a group size of two with a resulting simple dialog between two subjects might not have been sufficient for that purpose. Group sizes larger than three might have made group interactions too challengingly complex for subjects.

For each user story an estimate was provided about how many development days the implementation of the user story could be expected to take. The total number of estimated days to implement all user stories exceeded the available time in the release, requiring subjects to prioritize the user stories.

Subjects were asked to come up with the following deliverables:

1. The set of user stories to be implemented in the first iteration of the release
2. The set of user stories to be implemented in the remaining three iterations of the release
3. Justifications for the inclusion and exclusion of user stories
4. A list of questions to be clarified before the next meeting (including the clarification of ambiguities, contradictions and duplications in the set of user stories)

The manipulation involved the degree to which conceptual models were used to achieve this task[12]:

- Groups in the control condition received no conceptual models and no instructions about the creation or use of conceptual models.
- Groups in treatment condition 1 received an initial conceptual model about the domain, but no specific instructions about how to use it.
- Groups in treatment condition 2 received the same initial conceptual model, and were instructed to use it to perform their task. In particular, they were asked to correct/extend

[12] Note that this was the initial plan. After experiences from early sessions, a third type of treatment was added, for a total of four different conditions – one control and three treatment conditions (see Table 16).

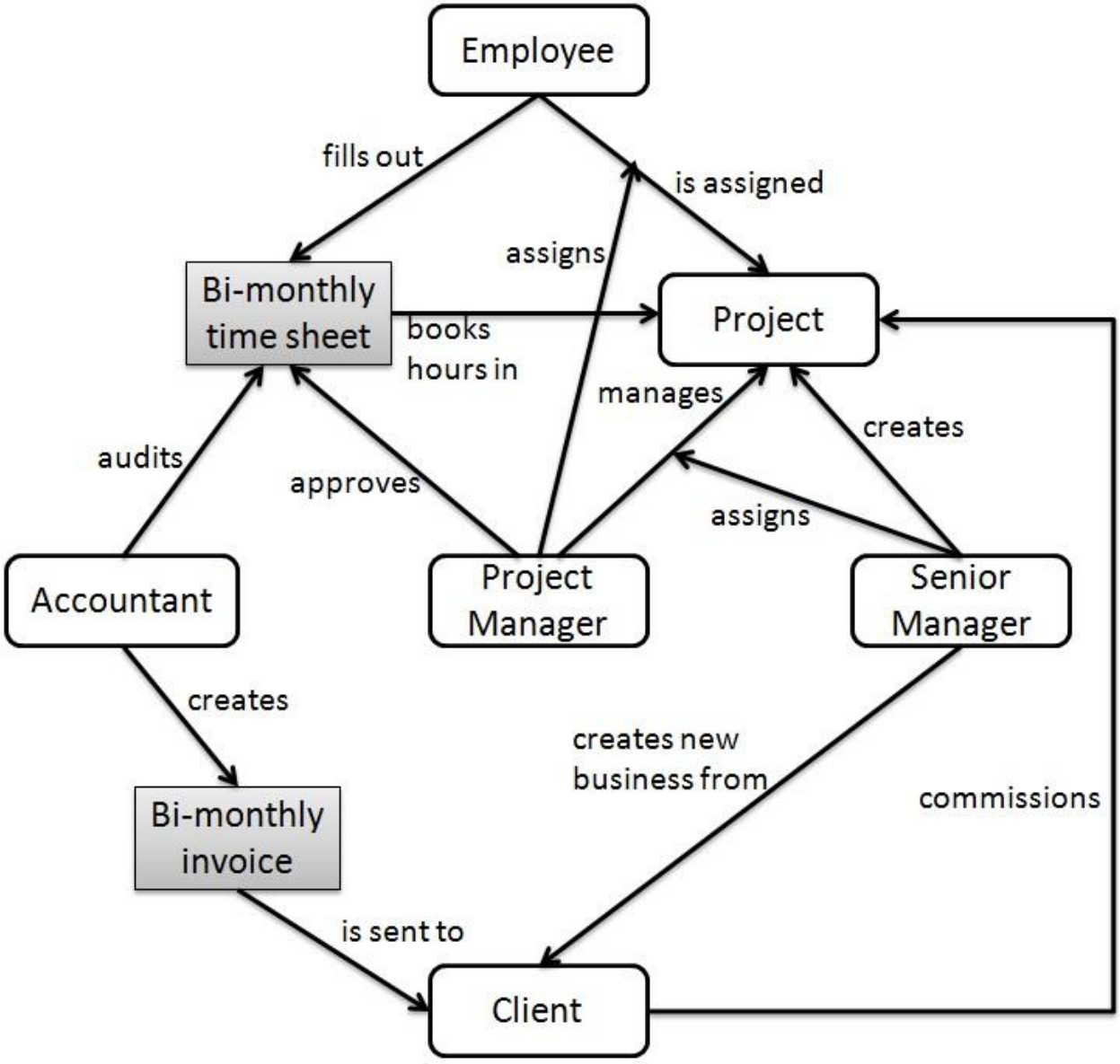
the model as needed to accurately reflect the current understanding about the domain at any given point.

The main considerations about the type of model (modeling grammar) provided to the treatment groups were that the model should be easy to understand and fit well to the type of system under development. To be an effective tool in an agile development setting, a model must be easy to understand, use and change without any specific technical background or training, because stakeholders of widely different technical skills should be able to use the model to communicate about the domain. Cognitive fit theory suggests that the type of the model should fit well with the type of system under development. Agarwal, Sinha et al. (1996) compared the use of object- and process oriented requirements modeling tools for object- and process oriented systems, and found that a match between tool and system lead to the best results.

The time reporting and billing system used in this experiment combines static and process aspects. The static component consists of different roles (such as accountants or project managers) and artefacts/concepts (such as time reports or projects), which could be well represented in simple entity-relationship or class diagrams. The process component contains information about how these roles cooperate to fulfil the business processes in the consulting company. Therefore, a combination of a static model (e.g., a class diagram) and a process model could be provided to the treatment groups. The downside of this approach would be the incurred complexity. Kim, Hahn et al. (2000) study how individuals cognitively integrate information from multiple diagrams of different types, and find that specific visual cues linking related elements in different diagrams improve the understanding of the domain by subjects. To avoid such complexity, rich pictures (Horan 2000), a simple yet flexible modeling grammar was chosen for the experiment. Rich pictures, introduced as part of the so-called soft systems

methodology (Checkland 2000), allow for models containing textual and graphical representations of any concepts of interest and their relationships in a domain. Figure 3 shows the rich picture provided to the treatment groups in the experiment. It shows the roles of interest (employee, accountant, project manager, senior manager, project and accountant) in rounded rectangles, while relevant artefacts (bi-monthly time sheets and invoices) are depicted as square rectangles. Relationships between the different roles and artefacts (typically given by actions, such as "fills out" or "audits") are shown with labeled arrows.

Figure 3: Rich Picture Provided to Treatment Groups



Each group of subjects is asked to perform two consecutive release planning meetings: The initial release planning meeting at the beginning of the project, and the second release planning meeting after the first release (to plan the second release). In a real project as specified above, four weeks would elapse between the two meetings. In the experiment, the second session took place one week after the first one, to account for the fact that subjects in the experiment have no exposure to the project between meetings. In a real project, such exposure (e.g., in the form of

emails with questions, weekly status reports etc.) would help individuals to remember more details from the previous meeting. The shortened elapsed time span should adjust for that difference.

Simulating two consecutive release planning sessions instead of just a single session significantly complicates the experiment in several ways. In particular, it becomes essential that all three subjects participate in both meetings. If one of the three subjects does not show up for the second meeting, it cannot take place, and the results of the incomplete session of the experiment become almost worthless[13]. Additionally, it becomes necessary to individually adjust the experimental materials for the second session based on the deliverables from the first session: Each group chose different user stories to include in the first release and to postpone to the second release. Since postponed user stories from the first release planning meeting become part of the backlog of user stories for the second release planning meeting, that backlog (along with the also provided list of already implemented user stories) had to be adjusted for each group.

The rationale behind the decision to simulate two consecutive release planning meetings despite these complications is that it allows subjects to get more familiar with each other, with the domain and system under construction, and with the task. Therefore, the results of especially the second session become much more realistic and meaningful, compared to an experimental setup in which only a single release planning meeting is simulated.

The sessions were recorded (by video). Therefore, in addition to the results of the task and the results of an exit survey at the end of each session (administered to each subject separately),

[13] Fortunately (and at least in part thanks to the compensation structure – see the next section), this did not happen. Only one subject had an unexpected scheduling conflict for the second meeting, but it was possible to reschedule the meeting for the whole group.

video recordings of the sessions are available. These videos can be used to study how subjects arrived at their deliverables.

The main experimental materials are included in Appendix D –Experimental Materials. Parts that are not necessary for an understanding of the experiment have been excluded, such as the recruitment letter and consent form, and the different variants of some of the documents for the different treatment conditions and roles (such as the task instructions and exit survey).

4.2.2 Experimental Sessions

4.2.2.1 Location and Structure

The experiment was performed in the MIS laboratory in the basement of the Henry Angus Building. The lab consists of an entrance room, a main experimental room and four small breakout rooms.

The experimental sessions were run by a paid research assistant who was not familiar with the research, in order to prevent the experimenter from accidentally influencing subjects. Each session had the following structure: When subjects arrived, they were asked to read and sign the consent form and then wait in the entrance room until all three subjects had arrived and done the same. Then subjects were handed the experimental materials and asked to individually familiarize themselves with the materials in three of the four breakout rooms. When all subjects were ready, the actual simulated meeting took place in the main experimental room. After the meeting, subjects were asked to fill out an exit survey – this was again done individually in three of the four breakout rooms. No time limit was imposed for any part of the sessions – subjects were instructed that the next step would commence once the preceding step was finished (e.g.,

that they were to fill out a short survey once they were finished with their meeting). The same procedure was repeated exactly one week later for the second session.

4.2.2.2 Sessions and Recruitment of Subjects

Subjects were recruited from the MIS classes COMM/BUSI 335 (Information Systems Technology and Development) in the previous summer term (July/August 2012) and the then current winter term (Sept-Dec. 2012). These courses provided subjects with the necessary background in IS development for the experiment. For participation, subjects received a compensation of $40 per session, and a bonus of an additional $20 for completing both sessions. In order to motivate subjects to take the task seriously, an additional bonus of $40 per subject for the top performing 1/3 of groups was offered. This relatively generous compensation with a bonus for completing both sessions was chosen because the attendance of all three subjects for both session was required for meaningful results.

For each of the three initially planned conditions (2 treatment conditions and one control condition) three groups of three subjects each were formed, for a total of 27 subjects. Subjects were put into groups according to their scheduling availability, and the groups were assigned to the three conditions by random draw, as was the assignment of subjects to roles within each group.

4.2.2.3 Changes Based on Early Sessions

Two pilot test sessions led to small refinements to the experimental materials, such as clarifications of the task instructions and user stories. Furthermore, in order to make it more easily possible to detect references to the provided conceptual model by subjects (in the treatment conditions), the model was taped onto the middle of the table at which the three

subjects were sitting in the simulated release planning meeting, where it was also easily in the reach of all three subjects. Figure 4 shows the experimental setting with an example screen capture of one of the treatment groups, with one of the subjects pointing at the provided model.

Figure 4: Sample Screen Capture of an Experimental Session

Finally, it turned out that (at least some) groups in the treatment conditions made very little use of the provided conceptual model. In particular, the difference in model use between the two initial treatment conditions was observed to be not very powerful – the groups specifically instructed to use the model did so not significantly more than the groups who were provided with the model without specific instructions about its use. Therefore, it was decided to add a new treatment condition 3 in which subjects were *not* provided with a conceptual model of the domain in the first session, but asked to create such a model based on the description of the domain and a simple example model from a different domain (see appendix, page 164). In the

second session, groups in this condition were given the same model as all treatment groups (with the explanation this was an updated version of the model they came up with in the first session, which incorporates what was learned about the domain during the first four iterations). The expectation was that having actively interacted with the model in the first session (by creating it) would make subjects more likely to go back to using the model in the second session.

Table 16 summarizes the resulting treatment and control conditions.

Table 16: Summary of Treatment/Control Conditions

Treatment/Control Condition	Description
Control	In both sessions: No conceptual model, no instructions about creating/using conceptual models
Treatment 1	In both sessions: Provided with a conceptual model of the domain, but no instructions about using it
Treatment 2	In both sessions: Provided with a conceptual model of the domain, and instructed to use it to perform the task, including to correct/extend the model as needed to reflect current understanding of the domain
Treatment 3	Session 1: Asked to create a conceptual model of the domain Session 2: Provided with a conceptual model of the domain, and instructed to use it to perform the task, including to correct/extend the model as needed to reflect current understanding of the domain

4.3 Results

As described in section 4.1.2, the focus of observation in the experiment lies primarily on the process of the task performance, and only in a secondary way on the product of the task, because

the former – due to its qualitative nature – can be expected to create much deeper insights in this exploratory research. Also, the small sample size (owed to the complexity of the experiment) makes the detection of statistically significant effects in quantitative data (the exit surveys and deliverables) unlikely.

Nevertheless, an analysis of the qualitative data might still be useful. Even the absence of statistically significant differences between the different treatment and control conditions might point at potentially valuable insights. For example, if the time taken to perform the task was not significantly different between the treatment and control conditions, then this would be some initial evidence that conceptual modeling practices need not strongly increase overhead. Therefore, the quantitative results are reported in Table 18.

Section 4.3.1 describes the quantitative results of the experiment, and sections 4.3.2 and 4.3.3 describe its main, qualitative results.

4.3.1 Quantitative Results

4.3.1.1 Quantitative Measures

The quantitative results of the experiment are based on exit surveys after each session, the deliverables of each session, and the measurement of time taken for the different parts of each session.

Table 17 gives an overview of the measures in the exit survey after session 1 (for the full measurement instrument, please refer to the appendix on page 158).

Table 17: Measures in Exit Survey - Session 1

Variable	Sample Item
Perceived difficulty of task	Performing the task required a lot of mental effort.
Confidence in results	I believe we came up with a good solution.
Domain understanding (perceived)	I believe I have a good understanding of the Time Reporting System.
Role identification (perceived)	I did a good job of fulfilling the specific responsibilities of the role that I played.
Difficulties encountered	My team encountered significant difficulties in performing its task.
Difficulties overcome	My team did a good job of overcoming any difficulties that we encountered.
Perceived early recognition	My team was able to understand dependencies between user stories well.
Perceived quick response	My team did a good job at prioritizing the user stories appropriately.
Perceived usefulness of diagram[14]	I found the provided diagram useful to understand the requirements.

All measures had three items which were measured on a 7-point Likert scale. Most variables should be fairly self-explanatory. The two variables "perceived early recognition" and "perceived quick response" link back to the measurement instrument for agility that was tested in the survey of practitioners (see section 3.1.2). They are the two items that most strongly correlated with self-reported agility. "Early recognition" describes how quickly the need for changes is recognized, and "quick response" refers to how quickly such recognized changes are actually acted upon.

[14] Only for treatment groups

The nature of the task implies that there is no single correct or optimal solution. Instead, the quality of the solutions the different groups came up with is subjective. Therefore, it was assessed independently by two expert raters, the author and another PhD student with significant experience in system analysis/design and teaching. When evaluating the solutions, the raters were unaware which treatment condition each solution belonged to. The results are on a scale from 0 to 20, with higher values indicating a better solution. In addition to these subjective measures, the time subjects took to familiarize themselves with the materials and to perform the simulated release planning meeting was recorded, as well as the number of times subjects used the provided domain model explicitly (i.e., by pointing at it and/or referring to it in the conversation). It should be pointed out that subjects in the treatment conditions received the conceptual model as a part of the materials that they received at the beginning of the session to study on their own. Since it is very likely that subjects looked at the model when they prepared for the simulated meeting, the model might still have had an impact on the understanding of the domain in the group even in cases where the group did not (or only very few times) explicitly refer to the model during the meeting.

In addition to these measures, the exit survey for session 2 measured actual domain understanding and actual role identification. These constructs were measured by multiple choice questions about the domain (e.g., "Who creates projects?") and short answer questions about the role each subject played (e.g., " What were the main responsibilities of the role you played in this experiment?").

Finally, the variables "early recognition" and "quick response" were measured in a more objective fashion than by simply asking subjects for their perception about both. For that purpose, the set of user stories provided as input for session 2 contained twelve dependencies,

contradictions and duplicates. An example dependency is that user story U32 (assigning budgets to tasks) requires the implementation of user story U31 (introduction of tasks to manage projects in a more fine-grained way) as a prerequisite. A sample contradiction is between U30 and U42 (they describe two alternative courses of actions when a project manager disagrees with the hours booked by an employee). U43 and U45 are an example of a duplication, as they both deal with giving individuals of roles other than "employee" the ability to book hours. For the full list of user stories and of dependencies, contradictions and duplicates, please refer to the appendix (page 156f.).

The dependencies, contradictions and duplicates formed the basis for measuring "actual early recognition": The more of them subjects overlooked (e.g., by not recognizing that two user stories contradict each other), the less well they seemed to be able to recognize the need for changes early. "Actual quick response" was measured similarly – the set of user stories included user stories that fundamentally changed the way the system was used (U35 – allowing employees to split time between different projects, or the combination of U31 and U33 – introducing tasks, which would implicitly allow subjects to book time on (tasks of) multiple projects). Not including either one of them significantly hampered the usability of the system. Therefore, if subjects failed to include one of these user stories in the first iteration, they were deemed to not respond to recognized (important) changes quickly.

Table 18 shows the quantitative results. The rows contain the measures (for measures in the exit survey, the mean of the responses for the three subjects per group), while the columns contain the different groups, or the mean of the results of groups per condition.

Table 18: Quantitative Results (Note: Results are not statistically significant, but still reported for completeness)

	Control				Treatment 1				Treatment 2 (T2)	Treatment 3 (T3)			Avg of	Avg of	
	Group 1	Group 3	Group 9	Avg	Group 4	Group 5	Group 8	Avg	Group 2	Group 6	Group 7	Avg	T2, T3	T1,T2,T3	(out of)
Session 1															
Perceived difficulty of task	3.33	3.33	3.56	**3.41**	3.67	5.33	3.56	**4.19**	3.89	3.22	3.56	**3.39**	**3.56**	**3.87**	7
Confidence in results	6.00	6.44	5.78	**6.07**	5.56	4.56	5.78	**5.30**	5.44	6.00	6.78	**6.39**	**6.07**	**5.69**	7
Domain understanding (perceived)	5.89	6.44	5.67	**6.00**	5.00	5.56	5.56	**5.37**	5.11	6.11	6.22	**6.17**	**5.81**	**5.59**	7
Role identification (perceived)	4.89	5.44	5.78	**5.37**	4.89	4.67	5.44	**5.00**	4.89	6.22	6.11	**6.17**	**5.74**	**5.37**	7
Difficulties encountered	4.33	3.67	3.67	**3.89**	2.33	4.00	3.67	**3.33**	3.00	4.67	2.67	**3.67**	**3.44**	**3.39**	7
Difficulties Overcome	5.50	6.00	5.67	**5.72**	5.00	5.33	5.33	**5.22**	5.67	6.00	6.67	**6.33**	**6.11**	**5.67**	7
Perceived early recognition	5.56	6.22	5.44	**5.74**	5.67	5.33	5.11	**5.37**	5.78	6.00	6.67	**6.33**	**6.15**	**5.76**	7
Perceived quick response	6.22	6.67	6.00	**6.30**	6.22	5.22	5.89	**5.78**	5.89	5.89	6.78	**6.33**	**6.19**	**5.98**	7
Perceived usefulness of diagram	n/a	n/a	n/a	n/a	5.56	5.56	4.78	**5.30**	5.44	4.78	5.78	**5.28**	**5.33**	**5.31**	7
Quality of solution - Rater 1	12.00	15.00	13.00	**13.33**	17.00	5.00	9.00	**10.33**	18.00	3.00	18.00	**10.50**	**13.00**	**11.67**	20
Quality of solution - Rater 2	14.00	12.00	14.00	**13.33**	12.00	6.00	12.00	**10.00**	18.00	8.00	16.00	**12.00**	**14.00**	**12.00**	20
Time - reading materials (minutes)	16	29	9	**18**	24	21	20	**22**	17	26	15	**21**	**19**	**21**	n/a
Time - session (minutes)	35	57	92	**61**	32	96	61	**63**	56	51	67	**59**	**58**	**61**	n/a
Number of references to model	n/a	n/a	n/a	n/a	1	8	3	**4**	*Recording failed*	3	1	**2**	**2**	**3**	n/a
Session 2															
Perceived difficulty of task	2.78	3.78	3.56	**3.37**	3.33	4.11	4.44	**3.96**	3.67	3.22	3.78	**3.50**	**3.56**	**3.76**	7
Confidence in results	5.89	5.67	5.33	**5.63**	6.44	5.33	6.11	**5.96**	5.89	6.00	6.56	**6.28**	**6.15**	**6.06**	7
Domain understanding (perceived)	6.56	6.00	5.22	**5.93**	5.44	5.22	6.11	**5.59**	5.56	6.33	6.33	**6.33**	**6.07**	**5.83**	7
Role identification (perceived)	6.00	5.67	5.89	**5.85**	4.67	5.33	5.89	**5.30**	5.67	6.44	6.56	**6.50**	**6.22**	**5.76**	7
Difficulties encountered	3.67	6.00	3.67	**4.44**	2.67	3.67	4.00	**3.44**	3.67	5.33	2.67	**4.00**	**3.89**	**3.67**	7
Difficulties Overcome	6.00	6.00	6.00	**6.00**	5.67	5.67	6.00	**5.78**	5.33	5.67	6.67	**6.17**	**5.89**	**5.83**	7
Perceived early recognition	5.56	6.00	6.22	**5.93**	6.22	5.33	6.00	**5.85**	6.00	6.22	6.89	**6.56**	**6.37**	**6.11**	7
Perceived quick response	6.22	6.11	6.00	**6.11**	6.67	5.44	5.78	**5.96**	5.67	6.33	6.89	**6.61**	**6.30**	**6.13**	7
Actual domain understanding	0.67	1.00	2.00	**1.22**	1.33	2.33	1.33	**1.67**	1.67	1.67	2.33	**2.00**	**1.89**	**1.78**	3
Actual role identification	2.00	2.67	2.67	**2.44**	2.33	3.00	3.00	**2.78**	2.67	2.00	3.00	**2.50**	**2.56**	**2.67**	3
Perceived usefulness of diagram	n/a	n/a	n/a	n/a	5.44	3.11	3.89	**4.15**	4.44	5.33	6.56	**5.94**	**5.44**	**4.80**	7
Early recognition -errors:	7	3	8	**6.00**	8	5	3	**5.33**	3	7	5	**6.00**	**5.00**	**5.17**	12
Quick response - errors:	1	1	0	**0.67**	1	0	1	**0.67**	1	0	0	**0.00**	**0.33**	**0.50**	1
Quality of solution - rater 1	6	11	10	**9.00**	11	10	15	**12.00**	14	8	15	**11.50**	**12.33**	**12.17**	20
Quality of solution - rater 2	9	13	12	**11.33**	8	15	13	**12.00**	13	13	15	**14.00**	**13.67**	**12.83**	20
Time - reading materials (minutes)	6	23	7	**12**	11	17	7	**12**	10	13	11	**12**	**11**	**12**	n/a
Time - session (minutes)	38	92	72	**67**	34	81	80	**65**	65	52	89	**71**	**69**	**67**	n/a
Number of references to model	n/a	n/a	n/a	n/a	2	0	0	**1**	1	1	13	**7**	**5**	**3**	n/a

4.3.1.2 Interpretation of Quantitative Results

The results show a high degree of variability between the different groups. For example, the duration of the first release planning meeting ranged between 35 and 92 minutes for the control condition. The fact that three groups of subjects (randomly assigned to the control condition from a relatively homogenous population of students) create such strongly different outcomes can at least in part be explained by the complexity of the task. As the recordings of the sessions reveal, subjects chose very different approaches to solve the task. Some groups went through all user stories together, dividing them into different categories according to their importance. Other groups split the user stories up and evaluated them individually. Some groups focused on user stories that can be postponed (and considered the rest to be important), while others looked for important ones and considered the rest candidates for postponement. Of the two groups that were asked to create a conceptual model in the first session (treatment condition 3), one group chose to create the model before evaluating the user stories, while the other group did the two tasks in reverse order.

In addition to the different ways in which groups as a whole address the task, individual differences between subjects also play an important role. For example, some groups made decisions very quickly and stuck with these decision – even if it they were only based on "a hunch". Other groups "agonized" over certain decisions by revisiting them again and again. The group dynamics were also different between groups – in some groups all three subjects participated essentially equally, while in other groups one subject was the clear "leader", or two subjects contributed more than the third subject.

In the face of such a high degree of variability, and with the small sample size of three groups per condition, even medium to large effect sizes in the variables of interest cannot be expected to

be detectable with statistical significance. To put this more directly – even if the use of conceptual models improved the performance of groups as measured for instance by shorter meeting times or fewer errors in the selection of user stories by say 20% (which would be a very worthwhile improvement in practice!), this effect would likely be hidden in the large variability within the three treatment conditions.

In light of this argument, the inconclusive quantitative results are not particularly surprising. For example, the average overall quality of the solutions is lowest in treatment condition 1 in session 1 and lowest in the control condition in session 2. Similarly, while in session 2 the mean error rates among the six treatment groups are lower than among the three control groups (5.17 and 0.50 vs. 6.00 and 0.56), this difference is not statistically significant due to the small sample size. Moreover, treatment condition 1 has fewer errors related to early recognition than treatment condition 3 (5.33 vs. 6.00), while the reverse is true for errors related to quick response (0.67 vs. 0.00). These results clearly do not justify firm conclusions.

On the other hand, it is an interesting finding that the addition of conceptual modeling did not cause detectable negative effects either. While the time taken for the simulated meeting by the individual groups varied strongly, the mean time for both the first and second sessions happened to be exactly the same for the 3 control and the 6 treatment groups – 61 minutes for session 1, and 67 minutes for session 2. Interestingly, the 2 groups that were asked to create a conceptual model of the domain in the first session (treatment condition 3) on average took less time to complete the meeting than the three groups in treatment condition 2 or the three control groups did (59 vs. 63 and 61 minutes). Therefore, it could be argued that the use of conceptual modeling practices might not cause significant overhead.

Another interesting observation is that the mean perceived domain understanding is slightly higher for the control groups than for the treatment groups (6.00 vs. 5.59 in session 1, and 5.93 vs. 5.83 in session 2), while mean actual domain understanding is noticeably higher in the treatment groups (1.78 vs. 1.22 on a scale from 0 to 3). Finally, while the perceived usefulness of the diagram is nearly the same between treatment condition 1 and treatment conditions 2 and 3 in session 1 (5.30 vs. 5.33), there are noteworthy differences in session 2. For treatment condition 3 it was noticeably higher than for the treatment condition 1: 5.94 compared to 4.15.

As discussed above, these observations are not firm conclusions, because the differences are not statistically significant due to the limited sample size and high degree of variability. However, they are still potentially interesting, for example as a basis for hypotheses for future studies.

4.3.2 Process Outcomes of the Treatment Groups

4.3.2.1 Analysis Approach

To analyze the recordings of the experimental sessions, several different theoretical frameworks were considered. Since the verbal communication about the domain between the subjects is the focus of analysis, a classification of different parts of communication, such as provided by Searle's speech act theory (Searle 1979), might be useful. Janson, Woo et al. (1993) apply a similar framework by Habermas to IS development. They classify speech acts into three categories: Instrumental (i.e., descriptive), strategic (i.e., with the intent to influence decisions of rational opponents) and communicative (i.e., with the intent to achieve understanding, agreement or consensus with others). A different approach based on Bera (2012) would be to specifically focus on communication breakdowns, which can be defined as "difficulties that occur during the process of problem solving". Specifically, an analysis of the recordings could consist of a

complete identification of all the speech acts used and the breakdowns that occur in all experimental sessions with the focus on differences between the various treatment and control conditions.

Unfortunately, the large variance between the different groups (even within the same condition) makes this approach impractical. For example, if a different number of breakdowns occurred between groups in different treatment conditions, this might be just as much a result of differences in how the different groups approached the problem as it is a result of the different treatment conditions.

Therefore, the analysis of the video recordings of the experimental sessions was performed as follows: First, the complete recordings of all treatment groups (including all three treatment conditions) were watched and the exact times of references to the conceptual model were written down. References could be either verbal, through a gesture (pointing), or a combination of both. Then, the exact conversation at the time of the use, together with a description of the model use (e.g., "project manager points to the bottom half of the diagram") was transcribed. In a final step the occurrences were grouped according to similarity and categorized. An approach in which the categories are not defined in advance, but during analysis of qualitative data is appropriate in situations where "existing theory or research literature on a phenomenon is limited" (Hsieh and Shannon 2005). Sarker (2009) uses the same approach in a study of agility in distributed IS development, and argues that "unearthing and refining concepts through constant comparison" can be used to "build and substantiate the emerging categories". Such a process in which the creation of categories or codes is done during the analysis of the data, rather than defining categories based on existing theories before coding begins is often referred to as open coding (see e.g., Flick, von Kardorff et al. (2004)).

At the top level, this approach yielded two different types of occurrences: Firstly, subjects discussing strategies about how to use the model (to solve their task), and secondly, subjects actually using the model for specific purposes without explicitly reflecting about their use of the model. For the first top-level category, referred to as "model use strategies", three sub-categories were identified, while the second top-level category "actual model use" was found to have five sub-categories. Table 19 lists all eight categories found in the six treatment groups, and the criteria for classifying individual occurrences of model use into them.

Table 19: Criteria for Classification of Model Use Occurrences (All Treatment Conditions)

Category	Description/Criteria for Categorization	Frequency
Model Use Strategies	**Discussions about the use of the model to solve the task**	**Total: 8**
Strategies for User Story Selection	Conversations that discuss how to pick user stories in a way that involves use of the model	4
Tracking the Completeness of the System	Conversations suggesting the use of the model *during the task* to explicitly decide whether the selected set of user stories is sensible/consistent/complete	2
Verifying Completeness of the System Post-Hoc	Conversations suggesting the use of the model *after the completion of the task* to explicitly decide whether the selected set of user stories is sensible/consistent/complete	2
Actual Model Use	**Actual uses of the model during the task performance**	**Total: 15**
Clarification of Specific Questions about the Domain	Situations in which the model is used to clarify a specific question about the domain / to find out a specific fact about the domain	4
Prioritization of User Stories	Situations in which the prioritization of user stories is discussed with explicit references to the model	4

Category	Description/Criteria for Categorization	Frequency
Understanding Dependencies between User Stories	Situations in which dependencies between user stories are discussed with explicit references to the model	3
Understanding how User Stories fit into the Context of the System as a Whole	Situations in which the model is used to describe or understand how a user story or set of user stories fits into the context of the system as a whole	2
Understanding how User Stories Change the System as a Whole	Situations in which the model is used to describe or understand how the inclusion of a user story or set of user stories changes the system as a whole	2

The following sections discuss all eight categories and provide example occurrences.

4.3.2.2 Strategies for User Story Selection

Subjects discussed strategies for using the model to determine which user stories should be included in or excluded from the upcoming iteration or release. One strategy to do this suggests as a criterion for inclusion of a new user story that it "branches out" from already implemented parts of the system:

Product Manager: "Because ideally, or not ideally, but it would make no sense to punch in something here [pointing at not-yet-implemented part of system]... so we want... we would want something related. So we can either branch out from here, here or here." (5-1: 38:20)[15]

[15] 5-1: 38:20 refers to: 38 minutes and 20 seconds into the recording of the first session of group 5. These numbers are not included to convey information to the reader, but to make keeping track of the locations in the videos easier.

A different strategy involves the determination of whether user stories cover functionalities described in the model as the key criterion for inclusion or exclusion in an iteration or release.

Project Manager [pointing at the diagram]: "Based on the roles, they should be able to do what they need to do, based on this diagram." (7-2: 12:40)

A different explanation for the same basic strategy:

Development Lead: "Look at each process, from each person and effects, and we'll try to look by each one [pointing at user stories] and see if [excluding] one of these [pointing at user stories] directly inhibits that, them doing that [pointing at diagram]. If it does, we will consider it a critical function." (7-2: 20:20).

A third explanation for the same strategy, using a specific example to illustrate:

Development Lead: "Given that they create projects [pointing to senior managers in the diagram], do any of these [pointing at user stories] facilitate that, and are necessary for that, and if so, then do that [i.e., include the user story in the iteration/release]." (7-2: 22:30)

4.3.2.3 Tracking the Completeness of the System

A different suggested use of the provided model consists of tracking how the consecutively selected user stories form a "complete" system that covers the key functionalities shown in the model provided.

One considered approach to achieve this revolves around "crossing off" (i.e., marking) links (i.e., functionalities) shown in the diagram as they are implemented by selected user stories:

Development Lead [pointing at the diagram]: "We can like, cross off the link or whatever." (7-2: 21:20)

In another group, essentially the same approach was suggested by re-drawing on scratch paper the parts of the model that are covered by already selected user stories:

Product Manager: "I'm re-creating what we have so far. We have these two [pointing at user stories], which is these... these lines. [pointing at re-drawn diagram]." (5-1: 38:00)

4.3.2.4 Verifying the Completeness of the System Post-Hoc

Similarly to the strategy in the previous section, some groups suggested to verify the completeness of the proposed system at the end of the sessions:

Product Manager [pointing at the diagram, discussing its use]: "[...] and then review the steps that we have taken, what we have taken out, and [...] ensure that what we want to do and what we don't want to do fits to our own goal." (6-1: 42:35)

A different group (that didn't use the model up to this point in the session) discusses the purpose behind the diagram provided:

Development Lead [pointing at diagram]: "We didn't quite use this, right?"
Project Manager: "No we didn't."
Development Lead: "So basically..."
Project Manager [pointing at diagram] "But I think... I think what this was for us to make sure that the system that we came up with did all of that. And I think it did." (8-1: 55:35)

4.3.2.5 Clarification of Specific Questions about the Domain

While the previous three sections described model use strategies (i.e., discussions about how to use the model), the remaining five describe actual model uses during the performance of the task.

The least complex actual use of the provided model consisted of improving the understanding of the domain by clarifying specific questions about it.

For instance, one group discussed the frequency of different tasks, including the completion of time sheets:

Project Manager [pointing at "bi-monthly time sheet" in diagram]: "But this is bi-monthly." (2-2 44:30)

In another example, the model is used to clarify the term "employee", as it is used in the user stories:

Development Lead: "So the senior manager would be an employee of the company?"
Product Manager [pointing at diagram]: "No, they're not. Employees are here. It's this role. And senior manager is this role. They are separated."
Development Lead: "So this is saying..."
Product Manager [pointing at diagram]: "Although senior manager is an employee of the company in the general sense, but in this case, it is not." (4-2: 7:30)

Another type of question about the domain involves understanding the responsibilities of the different roles:

Product Manager: "[...] because the accountant checks it."
Development Lead: "Does the accountant check it?"
Product Manager [pointing at diagram]: "Yeah, they have to audit the...[time sheets]." (5-1: 61:00)

Another example involves the differences between the roles project manager and senior manager:

Project Manager [pointing at diagram]: "The senior manager assigns... ahem he makes the projects, but he assigns project managers to the projects. And then the project manager assigns employees." (5-1: 89:12)

4.3.2.6 Prioritization of User Stories

Since the task consisted of prioritizing user stories, it is not surprising that subjects used the diagram directly to help them perform this task. One of the specific ways to use the diagram for this purpose is to examine a single specific user story at a time. For instance, two different groups were considering the priority of the same user story that would allow the tracking of non-billable employee hours:

Project Manager [reads/summarizes user story]: "Tracking employee time, even if it is not spent on client projects."
Development Lead [pointing at diagram]: "Who is the person? Senior management?"
Project Manager: "I don't think that's relevant, because we just want to know how much time they put into each project [pointing do diagram], for which they can then bill the client." (7:2: 23:30)

The other group had the following discussion about the exact same user story:

Development Lead: "So it's not related to client projects, so I don't know if it's necessary to do."
Product Manager: "I don't think it's necessary."
Development Lead: "Ok. So..."
Product Manager: "[...] so it just says... it also records the time that you're slacking off?"

Development Lead: "Basically, they want also where we build all these things to go from this cycle [uncovers diagram and points to upper half] to this cycle [points to lower half], we're doing everything for the client. We track all the activities for the client. But this is saying, do the timesheets for other things, for our HR purposes too. But I don't know if that's needed right away. First complete the cycle to the client, right?" (4-2: 22:46)

A different approach to prioritize user stories with the help of the diagram is by starting with a key functionality from the diagram, and identifying which user stories need to be implemented for it:

Product Manager [pointing at diagram]: "Well I think the other really important one is to create the project in the first place. I think there was one [pointing at user stories] that was like 'create the project'. So then that's kind of like the enter hours screen we need the raw data for. "
Project Manager: "Ok, with all this [pointing at user stories], we actually have everything [pointing at diagram], if you think about it. " (5-1: 40:25)

A final use of diagrams for the prioritization of user stories is in using the diagram to articulate a justification for the selection of a specific user story:

Project Manager: "It affects the entire system."
Product Manager: "It affects all the other workflows."
Development Lead: "Not just the workflows, too, though. It affects the system, so how you program the screen. That's going to look different. It also affects the process, so how you as an employee or how you as a manager or how you as an accountant apparently now can enter your hours [pointing at these roles in the diagram]." (7-2: 62:30)

4.3.2.7 Understanding Dependencies Between User Stories

A common use of the diagram was to determine the dependencies between different parts of the system. In particular, the order in which different parts of the system need to be implemented (e.g., part A must be implemented before part B, because part B cannot be used without part A) was considered using the diagram.

Development Lead: "So then accountants should come after the employee. [...] Like you want to think about how the process is, right? So the first thing... from the diagram [points to the diagram]... first thing, the employee needs to enter hours before the accountant can approve anything, before the accountant can audit anything and before the project manager can approve anything." (5-1: 8:05)

A different dependency considered by a different group was between user stories describing functions involving senior managers and user stories about employee functions:

Product Manager: "Employee will be, oh no, Senior Manager will be the next step."
Development Lead: "Ok, yeah, cause that was the part where I wasn't sure we'd prioritize based on what. "
Product Manager [pointing at diagram]: "Because they assign... I mean, they assign the project to the project manager. If they don't, the project manager won't be qualified to do any work."
Project Manager: [pointing at diagram]: "Ok, so they decide the project, right?"
Development Lead: [pointing at diagram]: "So this is done, right? The employees are already entering their times?"
Product Manager: "[pointing at diagram]: Well, you have to have a project so that the employee will work, so I think the senior manager will go before employee."

Development Lead: "Ok." (4-1: 13:20)

A similar exchange to prioritize between user stories for project managers and system administrators:

Development Lead: "Should we have the project manager before the systems stuff?"
Product Manager: "I don't think so, because you have to create an account for the senior manager first so that they can assign stuff to the..."
Development Lead: "Ok, that makes sense. So then basically, what you're saying is, after those two are done, go from senior manager to project manager, then employee, and then accountant last?" [Product Manager saying "project manager", "employee" and "accountant" at the same time while pointing at these entities in the diagram] (4-1: 14:30)

4.3.2.8 Understanding How User Stories Fit into the Context of the System as a Whole

A fairly simple and straightforward use of the provided diagram was to illustrate/understand how a user story or a set of user stories fit into the context of the whole system:

Project Manager [pointing at left hand side of the diagram]: "So right now we would have that... actually all the left of it... with these three [user stories][pointing at user stories]." (5-1: 39:30)

Another example:

Project Manager: "I guess this [pointing at user story] would work out for this way where it goes like that [pointing at diagram]." (5-1: 40:10)

4.3.2.9 Understanding How User Stories Change the System as a Whole

The model of the domain can also be used to understand the implications of adding a specific user story to the system as a whole:

Project Manager: "So the senior managers want to also book their hours. And since they work too, so shouldn't they also have to book their hours into the timesheet?"
Development Lead: "If you look at the diagram [pointing at the diagram] it doesn't even say that."
Project Manager: "Yeah, we definitely will have to change the diagram. [...] But it changes the role of the senior manager in general." (7-2: 5:15)

In a different example, the group discusses the impact of a user story that would allow project managers to define and manage tasks within their projects:

Project Manager [pointing at diagram]: "Then we'll have to add tasks here."
Development Lead [pointing at diagram]: "You'd have to add tasks. Here. Because it's going to go from here, actually, here to here [pointing at various points in diagram]."
Project Manager [pointing at diagram]: "The employees would do tasks instead of projects."
Development Lead [pointing at diagram]: "And this would also be tied to tasks." (7-2: 25:50)

4.3.3 Process Observations in the Control Groups

The decision to perform this study as a true experiment (in which subjects are randomly assigned to treatment *and* control conditions) makes it possible to gain insights about conceptual model use in an agile setting not only by studying how the treatment groups use the conceptual model, but also by comparing the process of the task performance between treatment and control groups.

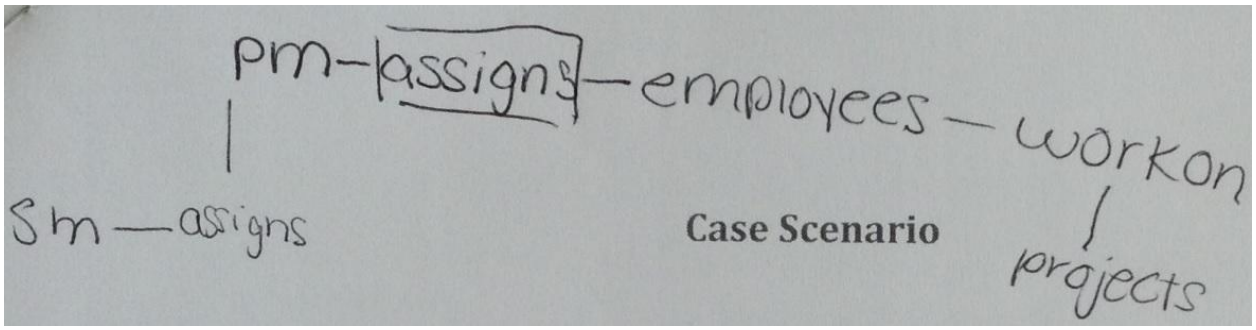
However, due to the high variance in task performance between the different groups discussed above (see section 4.3.1.2), the most direct approach for this, a one-to-one comparison of the behaviour of treatment and control groups, is not practical. Instead, the following sections describe noteworthy observations from the recordings of the control groups.

4.3.3.1 Ad-Hoc Use of Modeling

The maybe most surprising observation from the recordings is that all three groups in the control condition used some (at least rudimentary) conceptual modeling practices, even though they were not instructed to do so.

In the first meeting of group 1 and the second meeting of group 9, one subject each created a small diagram of the domain when they familiarized themselves with the experimental materials before the main experimental session. Since these materials were (consistent with the instructions subjects received) taken away from the subjects before the start of the simulated group meeting, subjects could not use these diagrams to communicate about the domain. Hence, the intended purpose of these diagrams was likely to improve own understanding of the domain. Figure 5 shows the simple domain model drawn by the project manager of group 1. It contains the roles senior manager (abbreviated as "sm"), project manager ("pm"), employee and project, and the relationships "assigns" (twice) and "works on".

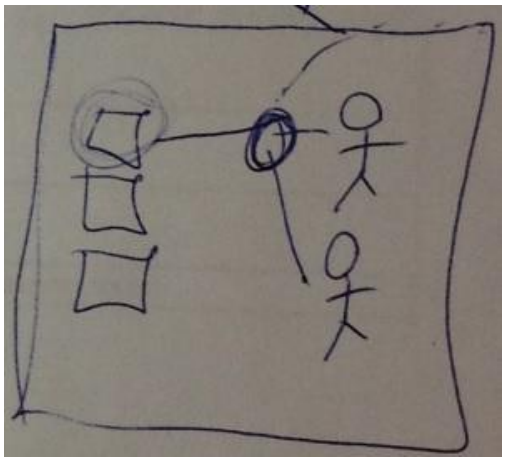
Figure 5: Subject-Drawn Mini-Model of the Domain



In the first meeting of group 9, the product manager drew a simple diagram (shown in Figure 6) illustrating the relationship between projects and employees, in the context of the following exchange:

Project Manager [pointing at different user stories]: "This would definitely be before that, because if no one is added, no employees are added, then they can't even go and do that, right?" Product Manager [after a 10 second pause]: "From my understanding, these projects, because the way the company works, it's like [begins drawing small diagram] you basically have like a group of projects, and a group of people that can do the projects. So like all this computer system [will] do is linking these people up, and making sure that all their hours for that specific project are being like calculated by the computer." (9-1:17:30)

Figure 6: Subject-Drawn Mini-Model of the Domain Used in the Session



Finally, in meeting 1, group 3 laid the user stories out on the desk, grouped by the role they are related to. They then used this layout like a large diagram when discussing the priority of the various groups of user stories:

Project Manager: "And now we go to employee."

Product Manager [pointing at groups of user stories]: "No, project manager. [...] [Employees] submit hours back to the project manager, which then go to the accountant." (3:1: 26:36)

4.3.3.2 Subjective Decision Making Without Explicit Solution Strategies

Compared to the treatment groups, which frequently discussed explicit strategies about how to prioritize user stories, control groups tended to make decisions without explicit justifications. Instead, the importance or urgency of the different user stories was assessed subjectively. The following exchanges in the second meeting of group 3 highlights this:

Project Manager: "I'm sorry, I'm getting really confused.[...] I'm really not sure – I don't really have a clear view of the priority of each user story." (3-2: 58:50)

Project Manager: "I don't really have that much confidence this time, you know. I really feel there is not too much logic here."

Development Manager: "I think those ones [the user stories] are for us to argue." (3-2: 64:35)

When control groups did attempt to come up with strategies for user story prioritization, they tended to struggle significantly more than treatment groups did. The following exchange in session 1 of group 1 (not fully transcribed for space reasons, but instead paraphrased and with the time in the session added) highlights this:

7:18: Project Manager: What is the priority – bottom up, employees entering hours, or top down, from project managers?

7:42: Development Lead: Suggests bottom up.

8:02: Project Manager: How can an employee enter hours if they don't have a project?

8:09: Product Manager: Just get the basics done in first iteration – more time in rest of release.

8:55: Project Manager: But do you guys understand what I mean, we need a direction, bottom up or top down?

[after a 30 second pause]: 9:35: Development Lead: Top down? Project Manager agrees.

9:50: Project Manager: "But I don't know."

9:57: General agreement to use top down.

One interpretation of this comparatively lengthy discussion and the uncertainty that subjects had in it (compared to for instance the exchanges in section 4.3.2.2) is that the absence of a conceptual model made the discussion more abstract (less grounded in the artifacts available), and therefore more cognitively challenging. Obviously, the limited data does not allow a firm conclusion; however, this might be an interesting aspect for future study.

4.4 Summary and Conclusions

In summary, subjects in the treatment groups used the model to support both their understanding of and their communication about the domain, as the literature about conceptual modeling would suggest: For instance, understanding can be improved when specific questions about the domain are answered by referring to the conceptual model, while communication is supported when subjects illustrate by means of pointing which parts of the model are affected by certain user stories.

The experiment also indicated that the use of conceptual modeling did not negatively impact agility. Neither did it increase the amount of time that groups needed to perform their task, nor was there any indication that it restricted the flexibility of the groups. Based on this, it can in fact be argued that model use even contributed to agility (see Table 20).

Table 20: Impact of Conceptual Model Use on Agility

Key Dimension of Agility	Impact of Conceptual Model Use
Early recognition of the need for changes	The use of the model to track the completeness of the system or to verify its completeness post-hoc (see sections 4.3.2.3 and 4.3.2.4) can help subjects to understand missing features, increasing the likelihood that the need to implement such missing features is recognized.
High degree of tangibility of intermediate results	The use of the model to understand how user stories fit into the context of, and how they change the system as a whole (see sections 4.3.2.8 and 4.3.2.9) can have a similar effect as an increase in tangibility, because it might allow certain predictions about how the system would behave if specific user stories were implemented (before they actually are implemented).
Quick response to recognized required changes	The use of the model to prioritize user stories, to understand dependencies between user stories and to select user stories strategically (see sections 4.3.2.6, 4.3.2.7 and 4.3.2.2) can support subjects in implementing the most urgent user stories first, which reduces the response time to critical changes.
Low overhead/leanness	Treatment groups did not take longer to complete their task than control groups.
Process agility	Treatment groups used the conceptual model in a wide range of different ways, indicating that model use did not significantly inhibit the ability to change development practices.

The analysis of the recordings of the treatment groups provided first insights into how conceptual modeling use might be beneficial in agile development. Specifically in the context of tasks

involving understanding of and communicating about requirements, the uses of the provided model by the treatment groups indicated that conceptual models might be useful to support

- a detailed understanding of the domain
- the prioritization of requirements
- the verification that a set of requirements is consistent/complete
- a detailed understanding of the dependencies between different requirements
- an understanding of how requirements fit into the context of the system as a whole, and how they impact the system as a whole.

These model uses can be the basis of hypotheses for future research. For instance, a more structured experiment (with consequently less variance) might focus on just one of the points above, with the goal to provide evidence that the use of conceptual models for these purposes does in fact improve the outcome by improving domain understanding or communication about the domain.

Chapter 5: Conclusion

This final chapter integrates the results of the thesis by providing a summary of its findings, highlighting its contributions to the literature, and discussing both limitations of this work and opportunities for future research.

5.1 Summary

This work addressed the role of conceptual modeling in agile IS development in three parts: It provided theoretical considerations about the role conceptual modeling might play in agile development, examined (by means of a survey of practitioners) which role conceptual modeling currently plays in actual agile development practice, and studied the specific impact of the use of conceptual modeling in an agile setting (using an experiment). In order to achieve these objectives, an operationalizable definition of agility – as used in this work, in the form of an outcome-oriented, development method independent construct – had to be developed.

The goal of conceptual modeling as defined by Mylopoulos (1992), the improvement of understanding of and communication about a domain, provides a strong argument for the usefulness of conceptual modeling in any information system development methodology. This is specifically true for agile development methods, where the scarcity of documentation in general might make models of the domain (which would be more stable than documentation about other aspects, such as the design or code of the system under development) particularly useful. Existing anecdotal evidence supports this (see section 2.3.3), especially since agile development methods tend to rely on significant amounts of informal communication (Rubin and Rubin (2010), Abrahamsson, Warsta et al. (2003)), which can be facilitated by suitable conceptual models.

In order to address the remaining two objectives, an operationalizable definition of agility and a measurement instrument based on it had to be developed. Existing instruments (e.g., Maruping, Venkatesh et al. (2009), Sidky and Arthur (2007)) measure the use of specific practices prescribed by existing agile development methods, rather than looking at agility in a method-independent way. The proposed definition of agility is based on five identified key dimensions of agility – early recognition of the need for changes, high degree of tangibility of intermediate results, quick response to recognized required changes, low overhead/leanness and process agility.

A survey of IS development practitioners was used to study to what degree conceptual modeling practices are actually used in agile development. The results supported the theoretical considerations above – the more agile respondents used a wider range of conceptual modeling practices than the less agile respondents. Additionally, higher agility was correlated with more effective communication about the domain, as the more agile respondents reported less frequent need to repeatedly inquire about or explain the same set of requirements. An additional result of the survey was that the measurement instrument based on the five key dimensions of agility correlated well with self-reported agility, which provided evidence for its reliability and validity.

Finally, an exploratory lab experiment was used to more closely investigate how the use of conceptual modeling might affect agile development practices. In the experiment, subjects were asked to prioritize requirements in a simulated IS development project. A detailed analysis of the recordings of the experimental sessions showed that subjects in the treatment groups used the provided conceptual model of the domain for various purposes. For example, they referred to the model to understand details of the domain, to verify that a set of requirements is consistent or complete, or to understand how specific requirements fit into the context of the system as a

whole. The results also indicated that the use of conceptual modeling seemed not to reduce agility. Conversely, it can be argued that the various uses of the provided model indicate that conceptual model use can in fact increase agility as measured by the five key dimensions defined above.

In summary, going back to the research questions posed in section 1.1 – what role does conceptual modeling play in current agile development practice, and what role should it play? – the results of the two empirical studies can be summarized by saying that agile developers do in fact use conceptual modeling practices (even though agile methods typically do not prescribe them), and that there is some initial evidence that this might in fact be beneficial, because the use of conceptual models can improve understanding of and communication about a domain, and thus ultimately even contribute to agility.

5.2 Contributions

The work described in this thesis makes a number of contributions:

- A summary and synthesis of important parts of various different bodies of literature, from relevant aspects of agile development methods in software engineering to conceptual modeling and domain understanding in the system analysis and design literature
- A development-method independent conceptualization of agility, and a validated measurement instrument of agility based on this conceptualization
- From a survey of practitioners, data about current conceptual modeling use in agile development practice and about issues regarding domain understanding and communication about the domain

- From an exploratory lab experiment, insights into *how* conceptual modeling practices might be beneficial in an agile context. The experiences from this complex experiment can be used for the design of future studies (see also section 5.3)

These contributions relate to several bodies of literature. The conceptualization of and measurement instrument for agility is useful for the software engineering literature, which still struggles with fundamental questions such as the definition and measurement of agility, and a systematic analysis of the various development methods used by practitioners (Soundararajan, Arthur et al. 2012). As discussed in section 1.2.2, the numerous measurement instruments that have been proposed in the past either assume the use of a specific agile development method, are not suitable for the context of a survey (for instance because of an excessive instrument size of hundreds of questions), or are not targeted at the level of a development project, but for instance at a development method in the abstract. The same can be said about the information systems literature, in which limited measurement instruments are still commonly used, such as in Maruping, Venkatesh et al. (2009), where "agile methodology use" is measured by testing for specific development practices such as pair programming and unit testing that are not inherently tied to agility.

In addition, the insights into conceptual model use in groups (irrespective of the agile development context) are a contribution to the conceptual modeling literature, in which the majority of studies in the past have focused on individual model use. The findings about conceptual modeling practices in agile development from the survey, as well as the hypotheses of how conceptual models might in fact contribute to agility that can be derived from the results of the experiment are also contributions to the growing literature on agile development.

This thesis and its results also have significant practical relevance. Many IS development projects still face serious challenges with regard to budget overruns, delays and incomplete features (StandishGroup 2001)[16], even though an increasing number of projects have adopted agile practices (West and Grant 2010). Given the large economic importance of IS development projects, even small improvements in their success rates would have an important impact on organizations. The results in this thesis provide first supporting evidence for the hypothesis that conceptual modeling practices might improve outcomes in agile development projects. Even more importantly for practitioners, the detailed observations of how subjects used the conceptual model in the experiment could give them insights into how to best improve their existing development methods by augmenting them with suitable conceptual modeling practices.

5.3 Limitations and Future Research

The main limitations of this work lie in the specific weaknesses of the research methods chosen. As discussed in section 2.4, the choice of the research method for an empirical study is always a trade-off between the strengths and weaknesses of different choices. For example, the decision to use the survey method to study current development practices allowed the collection of data from a significantly larger number of projects than for instance the case study method would have permitted. However, a number of case studies would have conversely provided a more in-depth view of projects, and consequently potentially yielded deeper insights. Similarly, the choice of an experiment for the last part of the thesis allowed for more control over the manipulation than for instance a set of field studies would have. However, realism and consequently generalizability would have been higher in field studies.

[16] There is an increasing amount of work that criticizes the methodology used by the Standish Group, see e.g. Eveleens and Verhoef (2010). However, the argument here does not rely on the precise numbers of the Standish report, but only on the (still valid) observation that a significant percentage of IS development projects is challenged.

These limitations provide a starting point for future research. As Mingers (2003) points out, information system research could benefit from more multi-method research. To improve the confidence in the results of our research, we should strive to study the same research questions using multiple different research methods. Therefore, a more in-depth study of conceptual modeling use in current IS development practice, for example by means of case studies, would be a worthwhile avenue of future research. This would allow the triangulation of the results of the survey described in chapter 3. The results of the survey, including the dimensions of agility most predictive of self-reported agility, could serve as a starting point for determining constructs of interest in such studies. To increase the confidence into the results about the impact of conceptual modeling use, field studies complementing the experiment described in chapter 4 would be desirable.

The experiences and results from the experiment can provide important information for the design of future studies. For example, the different types of model uses identified in Table 19 could serve as concepts to look for in qualitative data gathered in field studies. Future experiments about the same research question could concentrate on one or on a small number of these model uses (such as "tracking the completeness of the system" or "prioritization of user stories") and study them in-depth in a more structured and focused experimental setup. This would reduce the high degree of variance that occurred in the experiment described in this thesis, and made statistically significant quantitative effects impossible to detect.

Similarly, the quantitative results shown in Table 18 can serve as a basis for hypotheses for future experiments. For example, does the use or creation of a conceptual model in fact not increase the amount of time needed to complete a task? The relationship between perceived and

actual domain understanding – where the use of conceptual modeling seemed to decrease the former, but increase the latter – might also be a worthy subject for future related work.

Another interesting area for future related research would be to study different conceptual modeling practices in agile development in more detail. Going back to the framework of conceptual modeling research by Wand and Weber (2002), such research could focus on either one of the four research areas – conceptual modeling grammars, methods, scripts and context. For instance, a study could compare the use of different types of modeling grammars or methods, or focus on individual or task-related contextual factors. Such work could have particularly high practical relevance, because insights such as "grammar X is more suitable for agile development than grammar Y under the specific set of circumstances Z" would be directly actionable information for practitioners who attempt to optimize their development methodologies.

Furthermore, it would be interesting to extend the research into different kinds of systems projects. The operationalization of agility applies most directly to custom information system development projects, and the experiment simulated such a project as well. Extending the work to other types of projects, such as for instance customizations of large-scale enterprise resource planning (ERP) systems (e.g., SAP), would be another worthwhile direction for future research.

Bibliography

Abrahamsson, P., K. Conboy, et al. (2009). "'Lots done, more to do': the current state of agile systems development research." European Journal of Information Systems **18**(4): 281-284.

Abrahamsson, P., J. Warsta, et al. (2003). New Directions on Agile Methods: A Comparative Analysis. 25th International Conference on Software Engineering, IEEE Computer Society.

Agarwal, R., A. P. Sinha, et al. (1996). "Cognitive Fit in Requirements Modeling: A Study of Object and Process Methodologies." Journal of Management Information Systems **13**(2): 137-162.

Agerfalk, P. J., B. Fitzgerald, et al. (2009). "Introduction to the Special Issue--Flexible and Distributed Information Systems Development: State of the Art and Research Challenges." Information Systems Research **20**(3): 317-328.

Andersen, D. F., G. P. Richardson, et al. (1997). "Group Model Building: Adding More Science to the Craft." System Dynamics Review **13**(2): 187-201.

Aydin, M. N., F. Harmsen, et al. (2005). "On the Adaptation of an Agile Information Systems Development Method." Journal of Database Management **16**(4): 24-40.

Bagozzi, R. P., Y. Yi, et al. (1991). "Assessing Construct Validity in Organisational Research." Administrative Science Quarterly **36**(3): 421-458.

Balijepally, V., R. Mahapatra, et al. (2009). "Are Two Heads Better than One for Software Development? The Productivity Paradox of Pair Programming." MIS Quarterly **33**(1): 91-118.

Beck, K. (1999). "Embracing Change with Extreme Programming." IEEE Computer **32**(10): 70-77.

Beck, K., M. Beedle, et al. (2001). "Manifesto for Agile Software Development." Retrieved Nov. 13, 2011.

Benbasat, I., D. K. Goldstein, et al. (1987). "The Case Research Strategy in Studies of Information Systems." MIS Quarterly **11**(3): 369-386.

Bera, P. (2012). "Analyzing the Cognitive Difficulties for Developing and Using UML Class Diagrams for Domain Understanding." Journal of Database Management **23**(3): 1-29.

Bergman, M., K. Lyytinen, et al. (2007). "Boundary Objects in Design: An Ecological View of Design Artifacts." Journal of the Association for Information Systems **8**(11): 546-568.

Boehm, B. (2002). "Get Ready for Agile Methods, with Care." IEEE Computer **35**(1): 64-69.

Boehm, B. W. (1988). "A Spiral Model of Software Development and Enhancement." IEEE Computer: 61-72.

Boehm, B. W. (2006). A View of 20th and 21st Century Software Engineering. International Conference on Software Engineering. Shanghai, China**:** 12-29.

Buglione, L. (2011). Light Maturity Models (LMM): an Agile application. Profes 2011**:** 57-61.

Burton-Jones, A. and M. J. Gallivan (2007). "Towards a Deeper Understanding of System Usage in Organizations: A Multilevel Perspective." MIS Quarterly **31**(4): 657-679.

Cao, L. and B. Ramesh (2008). "Agile Requirements Engineering Practices: An Empirical Study." IEEE Software **25**(1): 60-67.

Ceravolo, P., E. Damiani, et al. (2003). A Ontology-based Process Modelling for XP. Tenth Asia-Pacific Software Engineering Conference.

Checkland, P. (2000). "Soft Systems Methodology: A Thirty Year Retrospective." Systems Research and Behaviorial Science **17**: 11-58.

Chen, P. P.-S. (1976). "The entity-relationship model—toward a unified view of data." ACM Transactions on Database Systems **1**(1): 9-36.

Chin, W. W. (2010). How toWrite Up and Report PLS Analyses. Handbook of Partial Least Squares: Concepts, Methods and Applications. V. E. Vinzi, W. W. Chin, J. Henseler and H. Wang. Heidelberg, Springer Verlag**:** 655-690.

Churchill, G. A. J. (1979). "A paradigm for developing better measures of marketing constructs." Journal of Marketing Research **16**: 64-73.

Cockburn, A. (2007). Agile Software Development: The Cooperative Game. Boston, Addison-Wesley.

Compeau, D., B. Marcolin, et al. (2012). "Generalizability of Information Systems Research Using Student Subjects--A Reflection on Our Practices and Recommendations for Future Research." Information Systems Research **23**(4): 1093-1109.

Conboy, K. (2009). "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development." Information Systems Research **20**(3): 329-354.

Conboy, K., S. Coyle, et al. (2011). "People over Process: Key Challenges in Agile Development." IEEE Software **28**(4): 48-57.

Conboy, K. and B. Fitzgerald (2004). Toward a Conceptual Framework of Agile Methods: A Study of Agility in Different Disciplines. ACM workshop on Interdisciplinary software engineering research.

Davies, I., P. Green, et al. (2006). "How do practitioners use conceptual modeling in practice?" Data & Knowledge Engineering **58**(3): 358-380.

Dawande, M., M. Johar, et al. (2008). "A Comparison of Pair Versus Solo Programming Under Different Objectives: An Analytical Approach." Information Systems Research **19**(1): 71-92.

Dennis, A. R. (2001). "Conducting Research in Information Systems." Communications of the Association for Information Systems **7**.

Diamantopoulos, A. and J. A. Siguaw (2006). "Formative Versus Reflective Indicators in Organizational Measure Development: A Comparison and Empirical Illustration." British Journal of Management **17**(4): 263-282.

Dobing, B. and J. Parsons (2006). "How UML is Used." Communications of the ACM **49**(5): 109-113.

Dyba, T. and T. Dingsoyr (2008). "Empirical studies of agile software development: A systematic review." Information and Software Technology **50**(9-10): 833-859.

Dyba, T. and T. Dingsoyr (2009). "What Do We Know about Agile Software Development?" IEEE Software **26**(5): 6-9.

Easterbrook, S., E. Yu, et al. (2005). Do viewpoints lead to better conceptual models? An exploratory case study. Requirements Engineering.

Eisenhardt, K. M. and M. E. Graebner (2007). "Theory Building from Cases: Opportunities and Challenges." Academy of Management Journal **50**(1): 25-32.

Erickson, J., K. Lyytinen, et al. (2005). "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research." Journal of Database Management **16**(4): 88-100.

Eveleens, J. L. and C. Verhoef (2010). "The Rise and Fall of the Chaos Report Figures." IEEE Software **27**(1): 30-36.

Fitzgerald, B., G. Hartnett, et al. (2006). "Customising agile methods to software practices at Intel Shannon." European Journal of Information Systems **15**(2): 200-213.

Flick, U., E. von Kardorff, et al. (2004). A Companion to Qualitative Research, Sage Publications.

Gemino, A. and Y. Wand (2004). "A framework for empirical evaluation of conceptual modeling techniques." Requirements Engineering **9**(4): 248-260.

Glinz, M. (2007). On Non-Functional Requirements. 15th IEEE International Requirements Engineering Conference

Guba, E. G. and Y. S. Lincoln (1994). Competing paradigms in qualitative research. Handbook of qualitative research 2**:** 163-194.

Haikara, J. (2007). Usability in Agile Software Development: Extending the Interaction Design Process with Personas Approach. XP 2007, Como, Italy, Springer.

Hansson, C., Y. Dittrich, et al. (2006). "How agile are industrial software development practices?" Journal of Systems and Software **79**(9): 1295-1311.

Henderson-Sellers, B. and M. K. Serour (2005). "Creating a Dual-Agility Method: The Value of Method Engineering." Journal of Database Management **16**(4): 1-23.

Hevner, A. R., S. T. March, et al. (2004). "Design Science in Information Systems Research." MIS Quarterly **28**(1): 75-105.

Highsmith, J. (2004). Agile Project Management. Boston, Addison-Wesley.

Highsmith, J. and C. Alistair (2001). "Agile Software Development: The Business of Innovation." IEEE Computer **34**(9): 120-122.

Horan, P. (2000). Using Rich Pictures in Information Systems Teaching. 1st International Conference on Systems Thinking in Management**:** 257-262.

Hsieh, H. F. and S. E. Shannon (2005). "Three approaches to qualitative content analysis." Qual Health Res **15**(9): 1277-1288.

Humphrey, W. S. (1988). "Characterizing the Software Process: A Maturity Framework." IEEE Software: 73-79.

Iversen, J. H., L. Mathiassen, et al. (2004). "Managing Risk in Software Process Improvement: An Action Research Approach." MIS Quarterly **28**(3): 395-433.

Jacobson, I., G. Booch, et al. (1999). The Unified Software Development Process. Reading, Mass., USA, Addison Wesley.

Janson, M. A., C. C. Woo, et al. (1993). "Information Systems Development and Communicative Action Theory." Information & Management **25**: 59-72.

Kettunen, P. (2012). "Systematizing Software-Development Agility: Toward an Enterprise Capability Improvement Framework." Journal of Enterprise Transformation **2**(2): 81-104.

Khatri, V., I. Vessey, et al. (2006). "Understanding Conceptual Schemas: Exploring the Role of Application and IS Domain Knowledge." Information Systems Research **17**(1): 81-99.

Kim, J., J. Hahn, et al. (2000). "How Do We Understand a System with (So) Many Diagrams? Cognitive Integration Processes in Diagrammatic Reasoning." Information Systems Research **11**(3): 284-303.

King, W. R. (2005). "External Valididy in IS Survey Research." Communications of the Association for Information Systems **16**: 880-894.

Kling, R. and R. Lamb (1999). "IT and Organizational Change in Digital Economies: A Socio-Technical Approach." Computers and Society **29**(3): 17-25.

Kovitz, B. (2003). "Hidden skills that support phased and agile requirements engineering." Requirements Engineering **8**(2): 135-141.

Kung, C. H. (1989). "Conceptual Modeling in the Context of Software Development." IEEE Transactions on Software Engineering **15**(10): 1176-1187.

Lappo, P. and H. C. T. Andrew (2004). Assessing Agility. XP 2004, Springer-Verlag.

Larman, C. (2004). Agile & Iterative Development: A Manager's Guide. Boston, Addison-Wesley.

Lee, G. and W. Xia (2010). "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility." MIS Quarterly **34**(1): 87-114.

Lindvall, M., Muthig, Dirk, DAgnino, Aldo, Wallin, Christina, Stupperich, Michael, Kiefer, David, May, John, Kaekoenen, Tuomo (2004). "Agile Software Development in Large Organizations." IEEE Computer **37**(12): 26-34.

Liu, J., Q. Wang, et al. (2010). "Application of Agile Requirement Engineering in Modest-Sized Information Systems Development." 207-210.

Lyytinen, K. and G. M. Rose (2006). "Information system development agility as organizational learning." European Journal of Information Systems **15**(2): 183-199.

Maruping, L. M., V. Venkatesh, et al. (2009). "A Control Theory Perspective on Agile Methodology Use and Changing User Requirements." Information Systems Research **20**(3): 377-399.

Mayer, R. E. (1989). "Models for Understanding." Review of Educational Research **59**(1): 43-64.

McGrath, J. E. (1981). "The Study of Research Choices and Dilemmas." American Behavioral Scientist **25**(2): 179-210.

Mingers, J. (2001). "Combining IS Research Methods: Towards a Pluralist Methodology." Information Systems Research **12**(3): 240-259.

Mingers, J. (2003). "The paucity of multimethod research: a review of the information systems literature." Information Systems Journal **13**: 233-249.

Moody, D. (2005). "Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions." Data & Knowledge Engineering **55**(3): 243-276.

Moore, G. C. and I. Benbasat (1991). "Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation." Information Systems Research **2**(3): 192-222.

Mylopoulos, J. (1992). Conceptual Modeling and Telos. Conceptual Modeling, Databases and CASE. P. a. Z. Loucopoulos, R. , John Wiley & Sons.

Naur, P. and B. Randell (1969). Software Engineering - Report on a Conference Sponsored by the NATO Science Committee.

Nuseibeh, B. and S. Easterbrook (2000). Requirements Engineering: A Roadmap. Future of Software Engineering. Limerick, Ireland.

Offen, R. (2002). "Domain Understanding is the Key to Successful System Development." Requirements Engineering **7**: 172-175.

Orlikowski, W. J. and C. S. Iacono (2001). "Research Commentary: Desperately Seeking the "IT" in IT Research—A Call to Theorizing the IT Artifact." Information Systems Research **12**(2): 121-134.

Orr, K. (2004). "Agile Requirements: Opportunity or Oxymoron?" IEEE Software **21**(3): 71-73.

Packlick, J. (2007). The Agile Maturity Map - A Goal Oriented Approach to Agile Improvement. Agile 2007.

Patel, C. and M. Ramachandran (2009). "Agile Maturity Model (AMM): A Software Process Improvement Framework for Agile Software Development Practices." International Journal of Software Engineering **2**(1): 3-28.

Qumer, A., Henderson-Sellers, B. (2008). "An evaluation of the degree of agility in six agile methods and its applicability for method engineering." Information and Software Technology **50**(4): 280-295.

Qumer, A., Henderson-Sellers, B. (2008). "A framework to support the evaluation, adoption and improvement of agile methods in practice." Journal of Systems and Software **81**(11): 1899-1919.

Ralph, P. (2010). Comparing Two Software Design Process Theories. DESRIST, Springer-Verlag.

Ramesh, B., L. Cao, et al. (2007). "Agile requirements engineering practices and challenges: an empirical study." Information Systems Journal **20**(5): 449-480.

Recker, J., M. Rosemann, et al. (2006). Extending the Scope of Representation Theory: A Review and a Proposed Research Model. 3rd ANU Information Systems Foundations Workshop, Canberra, Australia.

Ringle, C. M., S. Wende, et al. (2005). SmartPLS. Hamburg, Germany.

Rising, L. and N. S. Janoff (2000). "The Scrum Software Development Process for Small Teams." IEEE Software **17**(4).

Royce, W. W. (1970). Managing the Development of Large Software Systems. IEEE WESCON.

Rubin, E. and H. Rubin (2010). "Supporting agile software development through active documentation." Requirements Engineering **16**(2): 117-132.

Rumpe, B. (2002). Agile Modeling with the UML. RISSEF 2002, Springer-Verlag.

Sarker, S. (2009). "Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context." Information Systems Research **20**(3): 440-461.

Savolain, J., J. Kuusela, et al. (2010). "Transition to Agile Development - Rediscovery of Important Requirements Engineering Practices." 289-294.

Schleyer, T. K. L. and J. L. Forrest (2000). "Methods for the Design and Administration of Web-based Surveys." Journal of the American Medical Informatics Association **7**(4): 416-425.

Searle, J. R. (1979). A Taxonomy of Illocutionary Acts. Expression and Meaning - Studies in the Theory of Speech Acts, Cambridge University Press**:** 344-369.

Selic, B. (2009). "Agile Documentation, Anyone?" IEEE Software **26**(6): 11-12.

Seuffert, M. (2009). "Agile Karlskrona test." Retrieved January 14 2012, 2012, from http://www.piratson.se/archive/Agile_Karlskrona_Test.pdf

Siau, K. and M. Rossi (2011). "Evaluation techniques for systems analysis and design modelling methods - a review and comparative analysis." Information Systems Journal **21**(3): 249-268.

Sidky, A. and J. Arthur (2007). "A Disciplined Approach to Adopting Agile Practices: The Agile Adoption Framework." Innovations in Systems and Software Engineering **3**(3).

Soundararajan, S., J. D. Arthur, et al. (2012). A Methodology for Assessing Agile Software Development Methods. Agile 2012**:** 51-54.

StandishGroup (2001). Extreme Chaos, technical report.

Straub, D. W. (1989). "Validating Instruments in MIS Research." MIS Quarterly **13**(2): 147-169.

Suchman, L. (1995). "Making Work Visible." Communications of the ACM **38**(9): 56-64.

Talby, D., O. Hazzan, et al. (2006). Reflections on Reflection in Agile Software Development. Agile 2006.

Tichy, W. F. (2000). "Hints for Reviewing EmpiricalWork in Software Engineering." Empirical Software Engineering **5**: 309-312.

Turk, D., R. France, et al. (2005). "Assumptions Underlying Agile Software-Development Processes." Journal of Database Management **16**(4): 62-87.

Turk, D. E., L. R. Vijayasarathy, et al. (2003). The Value of Conceptual Modeling in Database Development: An Experimental Investigation. EMMSAD 2003.

Venkatesh, V., M. G. Morris, et al. (2003). "User Acceptance of Information Technology: Toward a Unified View." MIS Quarterly **27**(3): 425-478.

Vidgen, R. and X. Wang (2009). "Coevolving Systems and the Organization of Agile Software Development." Information Systems Research **20**(3): 355-376.

Vinzi, V. E., L. Trinchera, et al. (2010). PLS Path Modeling: From Foundations to Recent Developments and Open Issues for Model Assessment and Improvement. Handbook of Partial Least Squares: Concepts, Methods and Applications. V. E. Vinzi, W. W. Chin, J. Henseler and H. Wang. Heidelberg, Springer Verlag**:** 47-82.

Wand, Y. and R. Weber (1990). "On the deep structure of information systems." Information Systems Journal **5**: 203-223.

Wand, Y. and R. Weber (2002). "Research Commentary: Information Systems and Conceptual Modeling - A Research Agenda." Information Systems Research **13**(4): 363-376.

Ward, C. and L. Legorreta (2009). "Beyond Waterfall and Agile Methods: Towards a New Contingency Model for IT Project Management." Social Science Research Network.

Weber, R. (2003). "Still Desperately Seeking the IT Artifact." MIS Quarterly **27**(2): iii-xi.

West, D. and T. Grant (2010). Agile Development: Mainstream Adoption Has Changed Agility, Forrester Research Inc.

Williams, L., K. Rubin, et al. (2010). Driving Process Improvement via Comparative Agility Assessment. Agile 2010**:** 3-10.

Wohlin, C., M. Hörst, et al. (2003). Empirical Research Methods in Software Engineering. ESERNET 2001-2003, Springer Verlag.

Wynekoop, J. L. and N. L. Russo (1997). "Studying system development methodologies: an examination of research methods." Information Systems Journal **7**: 47-65.

Wyssusek, B. and J. M. Zaha (2007). Towards a Pragmatic Perspective on Requirements for Conceptual Modeling Methods. Workshop on Exploring Modeling Methods for Systems Analysis and Design, Trondheim, Norway.

Appendices

Appendix A – Survey Questions

Questions about Modeling Practices

Software developers use two basic types of models:

1) Conceptual Models, like flowcharts and free-form diagrams, represent the world, problem, or domain;

2) Design Models, like UML class diagrams, represent the software system.

Some models, like UML Use Cases, can do both. Models can have both text and graphics. In the following, we ask questions about the use of diagrams for both purposes – referring to them as ***conceptual models*** and ***design models***.

1. Which of the following modeling methods do you use for conceptual models?
   ( ) ER diagrams
   ( ) Class diagrams
   ( ) Use Case diagrams
   ( ) UML activity/sequence diagrams
   ( ) Process diagrams (e.g. BPMN/EPC)
   ( ) Other (please specify)

2. Which of the following modeling methods do you use for design models?
   ( ) ER diagrams
   ( ) Class diagrams
   ( ) Use Case diagrams
   ( ) UML activity/sequence diagrams
   ( ) Process diagrams (e.g. BPMN/EPC)
   ( ) Other (please specify)

3. Which media do you use to store/keep conceptual models?
   ( ) Computer files
   ( ) Paper
   ( ) Whiteboards
   ( ) Collaborative Websites (e.g. Wikis)
   ( ) Other (please specify)

4. Which media do you use to store/keep design models?
   ( ) Computer files
   ( ) Paper
   ( ) Whiteboards
   ( ) Collaborative Websites (e.g. Wikis)
   ( ) Other (please specify)

5. Which (if any) of the following software packages do you use for the creation of conceptual models?
   ( ) Word processor, e.g. Word
   ( ) Spreadsheet software, e.g. Excel
   ( ) Graphics package, e.g. Visio
   ( ) Tools in Integrated Development Environments, e.g. Rational or Eclipse
   ( ) Other (please specify)

6. Which (if any) of the following software packages do you use for the creation of design models?

( ) Word processor, e.g. Word
( ) Spreadsheet software, e.g. Excel
( ) Graphics package, e.g. Visio
( ) Tools in Integrated Development Environments, e.g. Rational
( ) Other (please specify)

7. What happens with conceptual models after they are created/used?

( ) They are typically discarded
( ) They are kept for personal future reference
( ) They are kept in a specific location for general team use (e.g. checked into the repository)
( ) Other (please specify)

8. What happens with design models after they are created/used?

( ) They are typically discarded
( ) They are kept for personal future reference
( ) They are kept in a specific location for general team use (e.g. checked into the repository)
( ) Other (please specify)

9. When you create conceptual models, do you use specific rules or guidelines?

( ) No specific rules – it is up to the individual
( ) There are rules about what modeling methods/media should be used
( ) There are rules about the situations in which models should be created
( ) There are rules about how to create models (e.g. about level of detail)
( ) Other (please specify)

10. When you create design models, do you use specific rules or guidelines?

( ) No specific rules – it is up to the individual
( ) There are rules about what modeling methods/media should be used
( ) There are rules about the situations in which models should be created
( ) There are rules about how to create models (e.g. about level of detail)
( ) Other (please specify)

Questions about Domain Understanding Issues and Model Use

1. How often do you find that you have to go back to users (or whoever else provides requirements to you, e.g. a product manager) for clarifications about requirements, even though the needed information was already provided previously, e.g. to another developer?

( ) Never
( ) Seldom
( ) Sometimes
( ) Often
( ) All the time

2. When the domain or requirements become more complex, how often do you...

	Never	Seldom	Sometimes	Often	All the time	N/a
use (additional) conceptual models?	_____	_____	_____	_____	_____	_____
make more changes to conceptual models?	_____	_____	_____	_____	_____	_____

3. If you use conceptual models: When the domain and requirements are very complex, what kind of changes to conceptual models do you do frequently (if any)?

( ) Additions of new concepts
( ) Deletions of concepts
( ) Changes to existing concepts

( ) Other (please specify)

4. How often do you use conceptual models as a communication tool when you need to explain parts of the domain or requirements to...

	Never	Seldom	Sometimes	Often	All the time
users?	_____	_____	_____	_____	_____
new team members?	_____	_____	_____	_____	_____
members of other teams?	_____	_____	_____	_____	_____
management?	_____	_____	_____	_____	_____

5. When you have to work on parts of the project with which you are not very familiar, how often do you use previously created conceptual models that may help you to understand the application domain and the requirements?

( ) Never
( ) Seldom
( ) Sometimes
( ) Often
( ) All the time

6. How often do you say something like "I have already explained this to you / another developer/ in a meeting"?

( ) Never
( ) Seldom
( ) Sometimes
( ) Often
( ) All the time

Questions about Demographic Information

The following questions are about the project about which you answered the questions above, i.e. the project that you currently spend most of your time on, or your last project if you are currently not working on a project. Please consider not only the coding part of the project, but the whole project from inception to completion.

1. How many months has the project been in progress? (This refers not only to coding, but to the whole project from inception to completion)

_____

2. How many more months is the project scheduled to last?

( ) Number of months:
( ) Project does not have a scheduled end date
( ) Don't know

3. How many people have been or are directly involved in the project?

Developers, full-time: _____
Developers, part-time: _____
Others (e.g. users, analysts), full-time: _____
Others (e.g. users, analysts), part-time: _____

4. Does your project use an approach based on one or several formal development methods?

( ) No specific method
( ) Method(s) used:

5. If you project is based on one or more specific formal development methods, how closely do you follow them?

( ) Very closely
( ) With few modifications
( ) With some modifications
( ) With significant modifications
( ) Mostly in name only

6. On a scale from 1 to 5, with 1 meaning "not agile at all" and 5 meaning "very agile", how agile would you say your project is, i.e. how quickly can you react to change?

( ) 1 - not agile at all
( ) 2
( ) 3
( ) 4
( ) 5 - very agile

Finally, a few demographic questions about your company and you. Again, all your answers in this survey will only be used in aggregation, and never to identify an individual.

7. How many employees does your company have approximately?

_____

8. What is your company's industry?

_____

9. What is the highest level of education you have attained?

_____

10. How many years have you worked in software development?

_____

11. How many years have you been in your current position?

_____

12. Which of the following describes your job best? Please check all that apply.

( ) Business Analyst
( ) Programmer/Software Engineer
( ) Team Lead
( ) Project Manager
( ) Quality Assurance
( ) Product Manager
( ) Other (please specify):

13. How many software development projects are you currently involved in (at any stage)?

_____

The final three questions are optional - please feel free to skip them if you are not comfortable answering them.

53. Which company do you work for?

_____

54. If you would like us to be able to contact you with follow-up research, please provide us with your contact information. (We will not use your contact information for any other purposes and never disclose it to third parties.)

Name _____
Email _____
Phone (optional) _____

55. Is there anything else you would like to tell us? Anything we should know? Any question you feel we should have asked, and didn't? Any issues with this survey?

_____
_____
_____
_____
_____

Appendix B – Measured Agility vs. Self-Reported Agility

Correlation between measured agility (AGILE_M) and self-reported agility (AGILE_S):

		AGILE_M	AGILE_S
AGILE_M	Pearson Correlation	1.00	.59
	Sig. (2-tailed)		.00
	N	120	115
AGILE_S	Pearson Correlation	.59	1.00
	Sig. (2-tailed)	.00	
	N	115	115

Analysis of variance between measured agility (AGILE_M) and self-reported agility:

ANOVA

		Sum of Squares	df	Mean Square	F	Significance
AGILE_M	Between Groups	20.58	4	5.15	19.16	.00
	Within Groups	29.54	110	.27		
	Total	50.12	114			

Analysis of variance between the five key components of agility (ER - Early Recognition, QR - Quick Response, TANG - Tangibility, LOWOH – Low Overhead/Leanness, PA – Process Agility) and self-reported agility:

ANOVA

		Sum of Squares	df	Mean Square	F	Significance
ER	Between Groups	46.26	4	11.56	8.75	.00
	Within Groups	145.42	110	1.32		
	Total	191.68	114			
QR	Between Groups	50.71	4	12.68	11.68	.00
	Within Groups	119.42	110	1.09		
	Total	170.13	114			
TANG	Between Groups	6.88	4	1.72	1.87	.12
	Within Groups	101.36	110	.92		
	Total	108.23	114			
LOWOH	Between Groups	3.77	4	.94	1.55	.19
	Within Groups	66.81	110	.61		
	Total	70.58	114			
PA	Between Groups	38.45	4	9.61	9.62	.00
	Within Groups	109.91	110	1.00		
	Total	148.35	114			

Appendix C – Survey Results about Modeling/Domain Understanding

Definition of Variables

Variable	Questions and Response Coding
ER	Early recognition (of the need for changes)
QR	Quick response (to recognized required changes)
TANG	Tangibility
LOWOH	Low Overhead
PA	Process Agility
CM_GrCount	Which of the following modeling methods do you use for conceptual models? Variable is count of the number of different types: ER diagrams, class diagrams, use case diagrams, activity/sequence diagrams, process diagrams, other
DM_GrCount	Same as above, except for design models (instead of conceptual models)
CM_Kept	What happens with conceptual models after they are created/used? They are typically discarded – 0, kept for personal future reference – 1, kept in specific location for general team use – 2
DM_Kept	Same as above, except for design models (instead of conceptual models)
CM_Rules	When you create conceptual models, do you use specific rules or guidelines? No specific rules – it is up to the individual – 0. For each of the following that is checked add 1: There are rules about what modeling methods/media should be used; There are rules about the situations in which models should be created; There are rules about how to create models (e.g., about level of detail); Other.
DM_Rules	Same as above, except for design models (instead of conceptual models)
RE_CLARIFY	How often do you find that you have to go back to users (or whoever else provides requirements to you, e.g., a product manager) for clarifications about requirements, even though the needed information was already provided previously, e.g., to another developer? (Likert scale from 1– almost never to 5 – almost always)
CM_ADD	When the domain or requirements become more complex, how often do you use (additional) conceptual models? (Likert scale from 1– almost never to 5 – almost always)
CM_CHANGE	When the domain or requirements become more complex, how often do you make more changes to conceptual models? (Likert scale from 1– almost never to 5 – almost always)
CHANGE_TYPES	If you use conceptual models: When the domain and requirements are very complex, what kind of changes to conceptual models do you do frequently (if any)? Variable is count of different kinds: Additions of new constructs, deletions of constructs, changes to existing constructs, other
COM_USERS	How often do you use conceptual models as a communication tool when you need to explain parts of the domain or requirements to users? (Likert scale from 1– almost never to 5 – almost always)
COM_TEAM	How often do you use conceptual models as a communication tool when you need to explain parts of the domain or requirements to new team members? (Likert scale from 1– almost never to 5 – almost always)
COM_OTH	How often do you use conceptual models as a communication tool when you need to explain parts of the domain or requirements to members of other teams? (Likert scale from 1– almost never to 5 – almost always)
COM_MA	How often do you use conceptual models as a communication tool when you need

Variable	Questions and Response Coding
	to explain parts of the domain or requirements to management? (Likert scale from 1– almost never to 5 – almost always)
COM_TOT	Mean of previous four variables
CM_USE	When you have to work on parts of the project with which you are not very familiar, how often do you use previously created conceptual models that may help you to understand the application domain and the requirements? (Likert scale from 1– almost never to 5 – almost always)
RE_EXPL	How often do you say something like "I have already explained this to you / another developer/ in a meeting"? (Likert scale from 1– almost never to 5 – almost always)
ADHERE	If you project is based on one or more specific formal development methods, how closely do you follow them? (Likert scale from 1– almost never to 5 – almost always)

Results – Descriptive Statistics

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean: Lower Bound	95% Confidence Interval for Mean: Upper Bound	Minimum	Maximum
CM_GrCount	HIGH	59	1.92	1.29	.17	1.58	2.25	0	5
	LOW	59	1.53	1.21	.16	1.21	1.84	0	5
	Total	118	1.72	1.26	.12	1.49	1.95	0	5
DM_GrCount	HIGH	59	1.71	1.11	.15	1.42	2.00	0	4
	LOW	59	1.64	1.20	.16	1.33	1.96	0	4
	Total	118	1.68	1.15	.11	1.47	1.89	0	4
CM_Kept	HIGH	59	1.47	.80	.10	1.27	1.68	0	2
	LOW	59	1.34	.84	.11	1.12	1.56	0	2
	Total	118	1.41	.82	.08	1.26	1.56	0	2
DM_Kept	HIGH	59	1.47	.80	.10	1.27	1.68	0	2
	LOW	59	1.37	.83	.11	1.16	1.59	0	2
	Total	118	1.42	.81	.07	1.28	1.57	0	2
CM_Rules	HIGH	59	.36	.76	.10	.16	.55	0	3
	LOW	59	.37	.64	.08	.21	.54	0	2
	Total	118	.36	.70	.06	.24	.49	0	3
DM_Rules	HIGH	59	.37	.74	.10	.18	.57	0	3
	LOW	59	.56	.93	.12	.32	.80	0	3
	Total	118	.47	.84	.08	.31	.62	0	3
RE_CLARIFY	HIGH	59	2.61	.97	.13	2.36	2.86	0	5
	LOW	59	3.07	1.27	.17	2.74	3.40	0	5
	Total	118	2.84	1.15	.11	2.63	3.05	0	5
CM_ADD	HIGH	59	2.81	1.11	.14	2.53	3.10	0	5
	LOW	59	2.56	1.44	.19	2.18	2.94	0	5
	Total	118	2.69	1.29	.12	2.45	2.92	0	5

		N	Mean	Std. Deviation	Std. Error	95% Confidence Interval for Mean: Lower Bound	95% Confidence Interval for Mean: Upper Bound	Minimum	Maximum
CM_CHANGE	HIGH	59	2.98	1.25	.16	2.66	3.31	0	5
	LOW	59	2.78	1.49	.19	2.39	3.17	0	5
	Total	118	2.88	1.37	.13	2.63	3.13	0	5
CHANGE_TYPES	HIGH	59	1.71	1.18	.15	1.41	2.02	0	3
	LOW	59	1.32	1.15	.15	1.02	1.62	0	3
	Total	118	1.52	1.17	.11	1.30	1.73	0	3
COM_USERS	HIGH	59	2.59	1.29	.17	2.26	2.93	0	5
	LOW	59	2.31	1.38	.18	1.95	2.66	0	5
	Total	118	2.45	1.34	.12	2.21	2.69	0	5
COM_TEAM	HIGH	59	3.14	1.53	.20	2.74	3.53	0	5
	LOW	59	2.98	1.37	.18	2.63	3.34	0	5
	Total	118	3.06	1.45	.13	2.80	3.32	0	5
COM_OTH	HIGH	59	2.76	1.43	.19	2.39	3.14	0	5
	LOW	59	2.37	1.40	.18	2.01	2.74	0	5
	Total	118	2.57	1.42	.13	2.31	2.83	0	5
COM_MA	HIGH	59	2.47	1.34	.17	2.12	2.82	0	5
	LOW	59	2.00	1.23	.16	1.68	2.32	0	5
	Total	118	2.24	1.31	.12	2.00	2.48	0	5
COM_TOT	HIGH	59	10.97	4.85	.63	9.70	12.23	0	20
	LOW	59	9.66	4.49	.58	8.49	10.83	0	18
	Total	118	10.31	4.70	.43	9.46	11.17	0	20
CM_USE	HIGH	59	2.78	1.30	.17	2.44	3.12	0	5
	LOW	59	2.66	1.35	.18	2.31	3.01	0	5
	Total	118	2.72	1.32	.12	2.48	2.96	0	5
RE_EXPL	HIGH	59	1.93	.83	.11	1.72	2.15	0	4
	LOW	59	2.63	1.16	.15	2.33	2.93	1	5
	Total	118	2.28	1.06	.10	2.09	2.47	0	5
ADHERE	HIGH	35	3.11	1.02	.17	2.76	3.47	1	5
	LOW	30	3.50	1.28	.23	3.02	3.98	1	5
	Total	65	3.29	1.16	.14	3.01	3.58	1	5

Results – ANOVA

		Sum of Squares	df	Mean Square	F	Significance
CM_GrCount	Between Groups	4.48	1	4.48	2.87	.09
	Within Groups	181.29	116	1.56		
	Total	185.77	117			
DM_GrCount	Between Groups	.14	1	.14	.10	.75
	Within Groups	155.63	116	1.34		
	Total	155.76	117			

		Sum of Squares	df	Mean Square	F	Significance
CM_Kept	Between Groups	.54	1	.54	.81	.37
	Within Groups	77.93	116	.67		
	Total	78.47	117			
DM_Kept	Between Groups	.31	1	.31	.46	.50
	Within Groups	76.51	116	.66		
	Total	76.81	117			
CM_Rules	Between Groups	.01	1	.01	.02	.90
	Within Groups	57.32	116	.49		
	Total	57.33	117			
DM_Rules	Between Groups	1.03	1	1.03	1.44	.23
	Within Groups	82.34	116	.71		
	Total	83.36	117			
RE_CLARIFY	Between Groups	6.18	1	6.18	4.85	.03
	Within Groups	147.76	116	1.27		
	Total	153.94	117			
CM_ADD	Between Groups	1.91	1	1.91	1.16	.28
	Within Groups	191.49	116	1.65		
	Total	193.40	117			
CM_CHANGE	Between Groups	1.22	1	1.22	.65	.42
	Within Groups	219.12	116	1.89		
	Total	220.34	117			
CHANGE_TYPES	Between Groups	4.48	1	4.48	3.31	.07
	Within Groups	156.98	116	1.35		
	Total	161.47	117			
COM_USERS	Between Groups	2.45	1	2.45	1.37	.24
	Within Groups	206.75	116	1.78		
	Total	209.19	117			
COM_TEAM	Between Groups	.69	1	.69	.33	.57
	Within Groups	243.90	116	2.10		
	Total	244.58	117			
COM_OTH	Between Groups	4.48	1	4.48	2.24	.14
	Within Groups	232.47	116	2.00		
	Total	236.96	117			
COM_MA	Between Groups	6.64	1	6.64	4.00	.05
	Within Groups	192.71	116	1.66		
	Total	199.36	117			
COM_TOT	Between Groups	50.25	1	50.25	2.30	.13
	Within Groups	2535.15	116	21.85		
	Total	2585.40	117			
CM_USE	Between Groups	.42	1	.42	.24	.63
	Within Groups	203.36	116	1.75		
	Total	203.77	117			
RE_EXPL	Between Groups	14.25	1	14.25	14.06	.00
	Within Groups	117.53	116	1.01		
	Total	131.77	117			

		Sum of Squares	df	Mean Square	F	Significance
ADHERE	Between Groups	2.40	1	2.40	1.82	.18
	Within Groups	83.04	63	1.32		
	Total	85.45	64			

Appendix D –Experimental Materials

Initial Case Description

You are working for the small software company SmallSoft, which creates custom software systems for its clients (usually small to mid-size companies). All the work SmallSoft does on behalf of its clients is performed in the context of projects. Each project has a project manager who is in charge of it, and a number of employees who are assigned to the project by the project manager. The company also has senior managers, who are responsible for creating new business in the form of new projects from existing or new clients and for assigning a project manager to each new project. All projects are done on a times-and-materials basis, i.e. clients are billed for the numbers of hours employees work on their projects (and for any other expenses that they might incur).

In the past, a paper-based system of time sheets was used to track the number of hours employees work on the various projects, and to bill clients accordingly on a bi-monthly basis. On the 15th and last day of each month, each employee would fill out a time sheet indicating the number of hours they worked on each day in the current pay period and submit it to their project manager for approval. After checking that the time sheets are ok, the projects managers would then forward them to accounting, who would in turn audit them, summarize all information and bill clients accordingly.

While this process worked reasonably well in the past when the company had only a few relatively small projects, its recent growth has made this method too cumbersome and error prone. Therefore, the decision was made to introduce a web-based system called TRS (*Time Reporting System*) to replace the paper-based workflow. To minimize risks and to be able to start using the system quickly, an agile development approach was chosen. In this approach, the whole system is implemented in a number of 1-week long iterations, each of which only implements a small part of the functionality expected from the complete system. Four consecutive iterations are grouped into a release. In other words, the then current version of the system is released to users every four weeks. Based on the feedback by users on each release the next release period is planned.

The desired functionality of the system is broken down into small chunks, each of which is specified by a "User Story", a 1-2 sentence long description of a feature that fits onto a index card. The amount of work it takes to implement a user story is estimated in developer days. A developer day is the amount of work a pair of developers can do in a single working day without external distractions. For quality reasons, all work is done in pairs ("four eyes see more than two"). The development team has 10 developers who form 5 pairs. It is assumed that they can spend 80% of their time working on the project – the remaining 20% are overhead (e.g. meetings, absences etc.). Therefore, in a 1-week iteration, the development team has a productivity of 5*4=20 developer days (5 pairs, each having 4 productive days per week), and consequently 80 developer days in a 4-iteration release.

Terminology

User Story:	A brief description of a requirement, used in the planning of releases and iterations. The time needed to implement a user story is measured in "developer days".
Iteration:	1-week long interval of time in which a small part of the whole functionality is implemented. It contains 20 developer days.
Release:	Four consecutive iterations that should result in a system with a useful subset of the

	complete functionality. It contains 80 developer days.
TRS:	Time Reporting System – the system under construction

Task Instructions – First Release Planning Meeting

Note: These example task instructions are for the first session, for the role development lead, in treatment condition 2.

The purpose of the upcoming meeting is to plan the upcoming first release of TRS, and the first iteration of that release. Specifically, in the meeting you will have to decide as a team which user stories should be implemented in this release and its first iteration. You should make sure that any critical users stories (i.e. user stories without which the usefulness or viability of the system as a whole is significantly impacted) are assigned to the first iteration, and that the user stories assigned to the whole release result in a system with a useful subset of the complete functionality.

As the **development lead** you are responsible for the correct implementation of the system. You represent the developers, and will have to make sure that their feedback is taken into account. This will be particularly important for the second meeting – since the project is just starting, there is no real feedback from the developers yet. You work together with the project manager (who is in charge of the project overall) and the product manager (who represents the users).

In the meeting, you will have the following materials available to perform this task:

- The case scenario (as contained in these materials)
- A diagram of the domain (as contained in these materials)
- The user stories (as contained in these materials) on index cards
- A deliverables template (on which the project manager will document the results of the meeting)

Please make use of the diagram provided when you perform your tasks, and update it (by corrections or extensions) as needed to accurately reflect your understanding of the requirements.

When you have familiarized yourself with the materials, please let the study assistant know, and hand these material back to her. Once all three participants are ready, the meeting can begin.

User Stories – First Release

This are the user stories for the first release – provided to subjects on index cards.

(The number in brackets behind each user story is the estimated number of developers days to implement it)

- (U01) All users of the system have to log into the system before they can use it, using individual user names and passwords. (4)
- (U02) When system administrators log into the system, they can choose between four screens: A "create new user screen", a "list all users screen", a "backup data screen" and a "restore data screen". (5)
- (U03) On the "create new user screen", system administrators can use an option to create new user accounts, which automatically assigns an initial passwords to them. (4)
- (U04) The "list all users screen" shows a list of all users of the system. (2)
- (U05) On the "list all users screen", system administrators can reset the password of individual users to a new initial password. (3)
- (U06) On the "backup data screen", system administrators can manually trigger a backup of the database with all user, project and time information. (3)
- (U07) On the "backup data screen", system administrators can create an automatic backup schedule of the database with all user, project and time information. (4)
- (U08) On the "restore data screen", system administrators can manually restore the whole database content to an earlier backup, which overwrites the current database content. (3)
- (U09) On the "restore data screen", system administrators can manually restore individual tables or records of the database content to an earlier backup, without overwriting the current database content. (6)
- (U10) When they log into the system for the first time, users will be forced to change their initial password. (3)
- (U11) When an employee logs into the system, the name of their current project (if any) is displayed on the top of the screen, and they choose between two screens: A "review past periods screen" and a "enter hours screen". (3)
- (U12) On the "review past periods screen", employees can review the hours they entered onto their project(s) on each day of each pay period in the last 12 months. (3)
- (U13) On the "enter hours screen", employees can enter the number of hours they worked on their current project on each day of the current pay period. (4)
- (U14) If an employee tries to submit more than 10 hours for any working day, or more than 60 hours for any week, the system should display an error message and force the user to correct the hours before they can be saved. (2)

- (U15) There is a "submit" button at the bottom of the "enter hours screen". When this button is clicked by an employee, a message box asking whether they are sure pops up. Once they confirm this, the hours are submitted to their project manager for review, and cannot be changed any more by the employee. (3)
- (U16) When a project manager logs into the system, they can choose between three screens: A "my hours screen" with which they can enter the hours they worked themselves on their current project, a "project management screen", and a "time report review screen". (4)
- (U17) The "project management screen" allows project managers to add available employees to their project, and to remove employees that no longer are on the project. (3)
- (U18) The "time report review screen" shows the hours that all employees of the project submitted for the current and last pay period. (3)
- (U19) The "time report review screen" allows project managers to approve the submitted hours of their employees for a just completed time period, which automatically forwards the hours to accounting. (3)
- (U20) When a senior manager logs into the system, they can choose between two screens: A "create project screen" and a "assign project manager screen". (3)
- (U21) On the "create project screen", senior managers can create new projects by entering all relevant data (such as project name, customer name etc.). (3)
- (U22) On the "assign project manager screen", senior managers can choose an available project manager (i.e. one who is currently not assigned to a project) from a list and assign him or her to a project as project manager. (3)
- (U23) When accountants log into the system, they are presented with a list of all current projects from which they can jump to a "project detail view" with detail information about each individual project. (2)
- (U24) On the "project detail view", accountants can choose a time report period and review the hours that individual employees entered for the project in that pay period. (3)
- (U25) On the "project detail view", accountants can view the total number of hours that have been booked on the project in a given pay period. (2)
- (U26) Set up technical infrastructure – web server. This has to be done before any features can be implemented. (5)
- (U27) Set up technical infrastructure – web framework. This has to be done before any features can be implemented. (5)
- (U28) Set up technical infrastructure – database. This has to be done before any features can be implemented. (5)
- (U29) Set up technical infrastructure – test infrastructure. This has to be done before any features can be implemented. (5)

Deliverables Template

This is the deliverables template for the first release planning meeting.

User stories for first iteration

User Story #	Estimated time to implement (days)	Justification/explanation for inclusion in first iteration
TOTAL:		(must be less than or equal to 20 developer days)

User stories for upcoming release(in addition to the ones shown above for the first iteration)

User Story #	Estimated time to implement (days)	Comments (e.g. modifications to user stories)

TOTAL:		(must be less than or equal to 60 developer days)

Remaining backlog in user stories (in addition to the user stories above)

User Story #	Estimated time to implement (days)	Justification/explanation for exclusion from release

Comments/questions to clarify before next meeting (e.g. ambiguities, contradictions, or duplications in user stories) (continue on back if more space is needed)

_____

_____

_____

_____

_____

_____

_____

Task Instructions – Second Release Planning Meeting

Note: These task instructions are for the role development lead, in treatment condition 2.

Task

The purpose of this meeting is to plan the upcoming second release and its first iteration of the TRS system. Specifically, you have to decide which user stories should be implemented in this release and its first iteration. You should make sure that any critical users stories (i.e. user stories without which the usefulness or viability of the system as a whole is significantly impacted) are assigned to the first iteration, and that the user stories assigned to the whole release result in a system with a useful subset of the complete functionality.

Your Role

As the **development lead** you are responsible for the correct implementation of the system. You represent the developers, and have to make sure that their feedback is taken into account. You work together with the project manager (who is in charge of the project overall, and responsible for creating the deliverables from this session) and the product manager (who represents the users). The project manager will have received additional feedback from senior management in the form of new user stories, while the product manager will bring new user stories that arose from users testing the previous release.

Feedback from your Developers

During the first release, it turned out that the estimates of the user stories were too optimistic. Before the first release, it was estimated that your developers could implement 20 developer days worth of user stories per week (iteration), resulting in 80 developer days worth of user stories in four weeks (release). Experience from the first release shows that actual productivity is about 25% below that. In other words, please make sure that for the second release only user stories totaling up to 15 developer days are scheduled per iteration, and up to 60 developer days per release.

As a consequence of this lower actual productivity, some of the user stories that were scheduled for the first release could not be implemented yet, namely U04, U05, U10.

This is in addition to the user stories that were postponed to a later release in the first release planning meeting (U06, U07, U08, U09, U12, U14)

In addition to this, developers came up with two additional user stories (U42 and U43)

All these user stories are provided on index cards as input for this upcoming release planning meeting. Note that the number in brackets on each user story represents the estimated number of developer days to implement it.

Please make use of the diagram provided when you perform your tasks, and update it (by corrections or extensions) as needed to accurately reflect your understanding of the requirements.

User Stories – Second Release

The user stories were provided on index cards.

Change Requests/feedback from Users (provided to Product Manager)

- (U30) If a project manager doesn't agree with an employee's hours on the "time report review screen", they can enter an explanation of the error or requested correction, which re-opens the hours for editing by the employee and notifies him/her by email. (4)
- (U31) Project managers would like to manage time on a more fine-grained level than projects. For that purpose, they want to be able to define tasks within each project on a new "manage tasks screen". (4)
- (U32) Project managers want to be able to manage tasks within their projects by assigning time budgets to them. (2)
- (U33) Employees should book their time on specific tasks within projects, not directly on projects. For that purpose, the "enter hours screen" should allow employees to select an unlimited number of tasks, and to enter hours into a separate row for each task. (4)
- (U34) The system should not allow tasks to be overbooked. In other words, if the hours an employee attempt to book on a task would exceed the time budget on that task, the system forces the employee to change his/her booking. (3)
- (U35) Most employees split their time between different projects. For that purpose, the "enter hours screen" needs to allow employees to enter hours into separate rows for each of the projects that they are assigned to – otherwise they can't use the system. (3)
- (U36) On the "project management screen", project managers want to see the hourly rate (i.e. the hourly rate billed to clients) of each employee before they add them to their project. (2)
- (U37) System administrators need a screen to set and change the hourly rate of employees. (4)
- (U38) Accountants want a screen that allows them to automatically create the invoice for a client for a time period. The calculation of this invoice needs to take the hourly rate of each employee into account. (5)
- (U39) Accountants would like to have a function to export hours to Excel. (3)
- (U40) System administrators would like to have a feature that allows them to distinguish between active and (temporarily) inactive users. (2)
- (U41) Accountants would like to have configurable deadlines for submission of time reports. For example, at the end of the quarter, it should be possible to require employees to submit their time reports two days earlier (based on best estimates) so that there is sufficient time for end-of-quarter reporting. (3)

Change Requests/feedback from Developers (provided to Development Lead)

- (Estimates-NotAUserStory) It turns out that the estimates for the time it would take to implement most user stories was too optimistic. For upcoming iterations, no more than 15 developer days should be scheduled, and for upcoming releases no more than 60. (0)
- (U42) It's not clear what should happen if a project manager doesn't approve the hours that an employee booked on their project. We suggest that the project manager can simply change the hours in that case. (3)
- (U43) In the current form employees belong to exactly one role. For example, senior managers or accountants have no screen to book their hours. We suggest to make the system more

flexible – each user can have several roles, and gets his/her screens accordingly. In particular, each user should have the role "employee" to be able to book hours. (6)

Change Requests/feedback from Projects Sponsor (Provided to Project Manager)

- (U44) Senior Management wants to use the system to track all employee time, even hours that are not spent on client projects (e.g. trainings, vacation, sick days, project acquisition activities etc.). Accountants should maintain such a list of general tasks in the system. (4)
- (U45) Senior managers should also have a screen that allows them to book their hours. (3)
- (U46) It should be possible to assign a project manager to manage more than one project. (2)
- (U47) It should be possible to assign more than one project manager to manage a project (e.g. so that they can share the workload, or that one can fill in when the other is absent). (2)
- (U48) Project managers should be able to define budgets for tasks. (2)
- (U49) The system should generate reminder emails when time reports are due or overdue. (3)
- (U50) Instead of global constraints (max. of 10 hours per day/ 60 hours per week), the maximum hours an employee can work varies between clients (depending on contracts) and should be configurable by the project manager. (4)
- (U51) Project managers should be able to configure whether employees can book time on Saturdays, Sundays and public holidays. (2)
- (U52) Management needs to be able to define an earlier deadline for the submission of time reports at the end of quarters. (3)

Dependencies, Contradictions and Duplications between User Stories

Dependencies between User Stories:

- U32 only with U31
- U33 only with U31
- U34 only with U33 and U32
- U44 only with U31
- U36 only with U37
- U38 only with U37

Contradictions between User Stories:

- U30 and U42
- U35 contradicted/superseded by U33
- U41 and U52 (who determines report deadline?)

Duplicates:

- U43 and U45
- U32 and U48
- U41 and U52

Post-Task Survey

These are the questions administered in the post-task survey of treatment group members after session 2. In the actual survey, the headings for the different sections were removed and the order of questions was scrambled. The control groups received the same survey, without the questions about the usefulness of the diagram

Please answer the questions below by circling a number.

[Perceived ease or difficulty of the task – adjusted fromMoore and Benbasat (1991)]

Performing the task required a lot of mental effort.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

The task we just performed was clear and understandable.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

Overall, I believe that the task I just performed was easy.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

[Confidence in results]

I'm confident in the results of our task.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

I think we did a good job with the task we were asked to do.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

I believe we came up with a good solution.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

[Domain Understanding] [perceived]

I believe I have a good understanding of the Time Reporting System.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

I think I have a good grasp of the requirements of the system and its context and goals.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

I understand the planned Time Reporting System and its requirements well.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

[Role Identification] [perceived]

I did a good job of fulfilling the specific responsibilities of the role that I played.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

I was able to identify well with the role that was assigned to me.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

I was comfortable with playing the role that I was asked to play in this session.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

[Perceived usefulness of diagram]

I found the provided diagram useful to understand the requirements.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

Using the diagram helped with the task.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

Without the diagram it would have been harder to understand the requirements and to perform the task.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

[Difficulties encountered]

My team encountered significant difficulties in performing its task.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

What, if any, are the main difficulties that your team encountered?

_____

_____

_____

[Difficulties overcome]

My team did a good job of overcoming any difficulties that we encountered.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

How did you overcome whatever difficulties you encountered?

_____

_____

_____

[Actual domain understanding]

Who creates projects? (Please check all that apply)

o Accountants

o Clients

o Employees

o Project Managers

o Senior Managers

Who approves the bi-monthly time sheets? (Please check all that apply)

o Accountants

o Clients

o Employees

o Project Managers

o Senior Managers

What activities are accountants responsible for? (Please check all that apply)

o Auditing bi-monthly time sheets

o Creating new business from clients

o Creating bi-monthly invoices

o Commissioning new projects

o Assigning employees to projects

[Actual role identification]

Which role did you play in this experiment?

_____

What were the main responsibilities of the role you played in this experiment?

_____

_____

Please name the two roles that the other two students played in this experiment.

_____

_____

[Perceived Early Recognition]

My team was able to understand dependencies between user stories well.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

My team was good at detecting potential contradictions between user stories.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

My team did a good job at handling ambiguities in the user stories.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

[Perceived Quick Response]

My team did a good job at prioritizing the user stories appropriately.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

My team was able to create releases with a useful functionality.

1	2	3	4	5	6	7
Strongly disagree						Strongly agree

My team managed to put the most urgent and important user stories in the next iteration or release.

1	2	3	4	5	6	7
Strongly disagree						Strongly

Additional Instructions for Treatment Condition 3 (with Sample Diagram)

Before you decide which user stories to assign to the first iteration and release, please **create a diagram** of the domain. Such a diagram should be useful as a communication tool throughout the whole lifecycle of the system. For example, if new team members join the development team, it can be used to familiarize them with the system and its requirements.

Such a diagram will be a "living document". In other words, there will be no "final version". Instead, it can be extended and corrected whenever new aspects about the domain or requirements are discovered or errors are found.

You can use whatever notation or symbols you find useful for your diagram. Below is a sample model from a university context – your diagram *could* look similar to it: