# Empirical Studies of Construction and Application of Use Case Models

Bente Anda

Thesis submitted for the degree of Dr.Scient.

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo

03.03.03

## Abstract

Requirements engineering is a critical part of the development of software systems. A plethora of techniques has been proposed to elicit and document the requirements of a software system. One technique that is now widely used in industry is use case modelling. Although the technique imposes no particular constraints on the methodology used in other phases in a development project, it is most frequently used in combination with object-oriented development. In addition to serving as a requirements capture vehicle and a means for developers to communicate with end users and customers, use cases are claimed to be useful for estimating software development effort and for facilitating the transition from functional requirements to software design.

Despite the important role of use case modelling in software development projects, and the numerous recommendations and claims that have been made about how to construct and apply use case models, there are very few systematic empirical studies in this field.

This thesis investigates the use case modelling process and the role of use cases in software development projects by means of a number of empirical studies. In total, five experiments with 37 professional software developers and approximately 250 students as subjects, three industrial case studies and 11 interviews with project managers and senior developers were conducted. The empirical studies were conducted on the construction of use case models by the use of guidelines and inspections, and the application of use case models in (a) the estimation of software development effort and (b) the design of object-oriented systems.

The results indicate that guidelines based on templates support the construction of use case models that are of higher quality, and that are easier to understand for the readers, than guidelines without specific details on how to document use cases. The results also indicate that quality may be further enhanced by combining the template guidelines with style guidelines for the documentation of the flow of events of each use case.

A taxonomy of defects in use case models, and a checklist-based inspection technique to detect such defects, were proposed. The evaluation of this technique shows that it may increase the detection of serious defects in a use case model and also that there is a large difference between the defects detected by the developers of the use case model and by the end-users of the resulting system.

The results further demonstrate that use case models can be used successfully to (a) estimate software development effort, (b) identify the prerequisites for successful use and (c) propose a refined and potentially improved version of an existing method for use case based estimation, the *use case points method*.

The results also indicate that the quality of a design model is affected by the way in which a use case model is applied in an object-oriented design process, in particular, that a process which applies a use case model in validation results in class diagrams that implement more of the requirements, while a use case driven process results in class diagrams with a better structure.

The empirical studies were exploratory because few similar studies have been conducted. Among other things, they required the development of original experimental designs. The major contribution of this thesis is, therefore, that it represents a starting point for more thorough empirical evaluation in the area of use case modelling

## Acknowledgements

First of all, I thank my supervisor Dag Sjøberg for his help, contributions, guidance and continuous enthusiasm. His indefatigable efforts have resulted in a stimulating research environment, which has made this period of my life unique and memorable.

I am also grateful to the other members of the Software Engineering Group at Simula Research Laboratory; Magne Jørgensen, Erik Arisholm, Amela Karahasanović, Marek Vokač, Hans Gallis and Kjetil Moløkken for interesting discussions and collaboration. I thank Magne Jørgensen in particular for his valuable comments, and a special thanks to Erik Arisholm and Amela Karahasanović for inspiration and for showing me that it was possible to finalize such a work.

I also thank Gunnar J. Carelius for providing all sorts of technical support, something that made it all a lot easier, and Tore Dybå for reviewing a part of this thesis.

I thank my M.Sc. students; Kirsten Ribu and Erik Syversen. Kirsten Ribu conducted and analyzed most of the interviews in Mogul and also contributed in other ways to the studies on use case based estimation. Erik Syversen contributed substantially to the initial design of the experiment on applying use case models in object-oriented design. I also thank all the students who participated in the experiments.

Important parts of the research conducted as part of this thesis would not have been possible without the cooperation from the company Mogul. This cooperation was organized as part of the industry-project PROFIT (PROcess improvement For the IT industry) funded by the Research Council of Norway. I therefore gratefully acknowledge the project manager of this project, Tor Ulsund, as well as Hege Dreiem and Endre Angelvik of Mogul. Thanks also to all the other people in Mogul who contributed data in several case studies and participated in interviews.

Thanks also to Reidar Conradi and Parastoo Mohagheghi for contributing to the studies on use case based estimation.

I also worked part-time at IBM for most of the time that I was working on my thesis. This provided me with valuable experiences with use case modelling both as a consultant and as a course instructor. I acknowledge Harald Eide and Tor Svelle for giving me this opportunity, and the participants in the courses on use case modelling who took part in one of the studies reported in this thesis.

Last but not least, thanks to my family for supporting and encouraging me, and to Martin and Simen in particular for making, this possible.

## Contents

The text formatting of the papers included in this thesis has been changed compared with the published versions of the papers to provide a common layout.

# Summary

## 1 Introduction

The IT industry is becoming one of the largest industries in the world. In industrialized countries, and increasingly in developing countries, computer systems are economically critical. The software in these systems represents a large and increasing proportion of the total system costs; in most industrialized countries, a large part of the Gross National Product is spent on software development. Modern economic and political systems depend, therefore, on our ability to produce software in a cost-effective way, and the choice of methods and tools for software development has a major economic impact.

Software engineering is concerned with the theories, methods and tools needed to develop software. Software engineering is cross disciplinary; it stretches from technical issues, such as databases and operating systems, to social and psychological issues because of the human intensive nature of software development.

The problems that software engineers are required to solve are often large and complex. Understanding the nature of a problem can be difficult. Software is intangible and immaterial; a software system can only be observed through the linguistic representations that we make of it or through the effects it produces when it is used. A software system is therefore difficult to conceptualize and communicate.

The development of large software systems involves many people, who have different requirements for the software system to be constructed and who will often not know exactly what they want from it. Consequently, it is often difficult to establish exactly what a software system should do. Yet, if it is not possible to define the requirements of a software system in a precise and comprehensible manner, it is difficult for the software engineers to know what to build, and it is likely that the system will be unsatisfactory.

The research presented in this thesis is carried out within the discipline of *requirements engineering*, which is a sub discipline of software engineering. The two major objectives of requirements engineering are to understand the problem that the intended system is supposed to solve, and to select and document the requirements on the system and its development.

Requirements engineering is a critical part of the software development process; it is claimed that more projects stumble or fail as a result of poor requirements handling than for any other reason (Hooper and Hsia, 1982, Glass 1998, Kulak and Guiney 2000), and that successful development of software systems depends on the quality of the requirements engineering processes (Regnell 1999). The requirements engineering process should create a solid basis for the remainder of the project.

A requirement may be functional, that is, it describes a system service or function. Alternatively, it may be non-functional, that is, it describes a constraint on the system or on the development process (Sommerville 2000). This thesis focuses on the elicitation, documentation and application of functional requirements in software development, and in particular on a specific technique for defining functional requirements called *use case modelling*.

The concept of use case modelling was introduced by Ivar Jacobson (Jacobson *et al*. 1992). He also introduced a *use case driven* approach to software development. Use case modelling has since emerged as one of the premier techniques for defining business processes and software systems. Use cases have also become an industry standard for defining requirements for the software systems created using present object-oriented development languages (Armour and Miller 2000), and use cases have become part of the Unified Modelling Language (UML) (OMG 2002).

Use case modelling is intended as a technique with many facets (Armour and Miller 2000), and could be used, among other things,

- as the requirements capture vehicle,
- as a means to communicate with end users and customers,
- to facilitate the transition from functional requirements to object and component structures, and
- to help estimate project size and required resources.

However, there is no commonly agreed theory on how to construct and apply use cases. Despite the apparent simplicity and widespread use of the use case modelling technique, many developers find it difficult to apply (Cockburn 2001) and need guidance (Weidenhaupt *et al*. 1998). This was also experienced by the author when working as a system designer, and as an instructor for courses on use case modelling, in IBM in the Scandinavian countries. There are many recommendations on how to construct and document use case models and on how to apply use case models in the development process, but very few empirical studies. It therefore appears that use case modelling, like the rest of the UML, has become a de facto standard without having been subjected to thorough evaluation (Briand *et al.* 1999a).

There is an increasing understanding in the software engineering community that new methods, techniques and tools should not just be suggested, published and marketed (Wohlin *et al*. 2000), but also evaluated through empirical studies (Basili *et al*. 1986, Basili *et al*. 1993, Rombach *et al*. 1993, Basili 1996, Tichy 1998, Zelkowitz

& Wallace 1998*)*. Such studies are vital for developing and validating our understanding of software engineering in general and the use case modelling technique in particular. In order to advance substantially our understanding of how use case models should be constructed and applied to improve the development process in the context of the organizations applying the technique, useful and reliable models of the use case modelling process must be developed.

This thesis investigates the role of use case modelling in requirements engineering by means of empirical studies, in the form of case studies in industry and controlled experiments with students as subjects. The thesis includes studies of both the construction and application of use case models. The results have implications for the successful application of requirements engineering, with use cases as an important basis for software development. The empirical studies conducted as part of this thesis also represent a starting point for more thorough empirical evaluation in the area of use case modelling.

## 1.1 Research Questions

The discussion above motivates the following overall research question:

*How can the use case modelling technique be enhanced to improve the software development process, particularly regarding the handling and application of functional requirements?*

Very few systematic empirical studies have been conducted on how to document and apply use case models in software development, and these are insufficient to form the basis for a theory on use case modelling or to provide advice on how to apply the technique in a specific context.

The activities related to the construction and application of a use case model in a software development project will affect each other. For example, the format of a use case model should be guided by characteristics of the development project (Cockburn 2001). The application of use case models in planning and development can also provide feedback on what information is necessary in a use case model to support those activities. The motivation for this thesis is:

1. Establishing a basis for empirical studies in the field of use case modelling by formulating testable questions that can be evaluated, and to initiate the conducting of such studies.
2. Investigating use case modelling from different viewpoints.

The research question is therefore refined into the following four questions:

1. How can guidelines for use case modelling enhance the quality and, in particular, the understandability of use case models?
2. How can an inspection technique tailored to the particular format of use case models facilitate the removal of defects and thereby enhance the quality of such models?
3. How can a system's use case model be applied to estimate software development effort?
4. How can a system's use case model be applied in an object-oriented design process?

## 1.2 Thesis Statement

Use case modelling is a common technique for handling functional requirements in object-oriented analysis. A use case model is also a primary artefact in several activities in a development project. Therefore, the construction and application of use case models have a significant impact on the development project especially in terms of quality and productivity, in particular

- The quality, in particular the understandability, of a use case model is enhanced by the use of appropriate guidelines.
- The quality, in terms of number of errors, of a use case model can be enhanced by the use of an appropriate inspection technique and by the participation of several kinds of stakeholders in the inspection.
- A use case model for a system can be used successfully to support the estimation of the effort needed to build the system.
- The quality of a UML class diagram can be improved by appropriate use of a use case model in making the transition from functional requirements to design model.

## 1.3 Contributions

The results from the studies conducted as part of this thesis go some way towards increasing our knowledge of how different methods for constructing use case models affect the quality of the use case model, and of how use case models can be applied in software development. This section provides a brief summary of the main contributions.

- Paper I shows how the quality of a use case model can be enhanced by means of the application of appropriate guidelines for its construction. The results from an experiment with students as subjects indicate that, when constructing use case models, guidelines based on templates are more useful than guidelines that contain no specific details on how to document use cases, because the resultant use case models are easier to understand. The results further indicate that the guidelines based on templates result in better use case models regarding other attributes concerning quality, such as correct actors and use cases and appropriate level of detail in the use case descriptions.
- Paper II propose a taxonomy of defects in use case models and an alternative technique for enhancing the quality of a use case model, a checklist-based inspection technique. The evaluation of this technique shows that it may increase the detection of serious defects in a use case model and also that there is a large difference between the defects detected by the developers of the use case model and those detected by the end-users of the resulting system. The results from the evaluation are used to suggest how the proposed inspection technique could be improved.

- Papers IV, V and VI demonstrate that use case models can be used successfully to (a) estimate software development effort, (b) identify the prerequisites for successful use, and (c) propose a refined and potentially improved version of an existing method for use case based estimation, the *use case points method*.
- Paper VII shows that the way in which a use case model is applied in an object-oriented development process may have an effect on the quality of the resulting UML class diagram. In particular, a use case driven process appears to lead to less creativity in the identification of classes and methods than does a process in which the use case model is instead used to *validate* the class diagram. A use case driven process may, however, lead to the class diagram having a better structure, in terms of higher coupling and lower cohesion, than may the alternative process.

The studies show that it is feasible to test empirically many of the claims about use case modelling and use case driven software development that are made in the literature. The empirical testing of these claims may represent one step on the way to building a theory of how different methods for use case modelling affect different aspects of a development project.

The focus of this work is on use case modelling which is a particular technique, but the questions related to how to document and apply functional requirements have a wider relevance and would, in the author's opinion, be applicable to other techniques for handling functional requirements.

Two of the studies conducted as part of this thesis were part of a software process improvement initiative in a software development company that focused on improving estimation practices (papers IV and VI), as well as part of a national project on software process improvement, PROFIT (PROcess improvement For the IT industry), funded by the Research Council of Norway. This work resulted in the company changing some of its estimation practices.

## 1.4 Overview of Thesis

This thesis is organized as follows:

**Summary**: This section (Section 1) is an introduction to the thesis. Section 2 presents the state of the art in use case modelling and use case driven software development, and also briefly describes the state of the art regarding software inspections and estimation of software development effort. Section 3 describes different research methods applied in empirical studies of software engineering and shows how different methods are used in the studies reported in this thesis. Section 4 summarizes the results of the research. Section 5 outlines possible directions for future work. Section 6 presents some concluding remarks.

**Papers**: This part includes the papers of this thesis. Table 1 gives an overview of the papers.

**Table 1. Overview of papers**

| Paper | Focus | Content |
|---|---|---|
| **I** | Constructing use case models using guidelines | Describes quality attributes of use case models and three different sets of guidelines for the construction of use case models. Describes an evaluation of the guidelines. |
| **II** | Validating use case models using an inspection technique | Proposes a taxonomy of defects in use case models and a tentative checklist-based inspection technique for use case models. Describes an evaluation of the inspection technique. |
| **III** | Obtaining a common understanding of the requirements | Discusses how results from research in cognitive psychology on how humans understand text can be used to increase our understanding of how use cases are read and understood by different stakeholders in a project. |
| **IV** | Estimating software development effort based on use cases | Describes the evaluation of a method for estimating software development effort, the use case points method, using three projects in a software development company. |
| **V** | As in paper IV | Describes the evaluation of the use case points method by comparing it with expert estimates. |
| **VI** | As in paper IV | Describes the prerequisites for applying the use case points method and how it can be tailored to a particular company.<br>Discusses challenges with use case modelling. |
| **VII** | Object-oriented design based on use cases | Describes an experiment conducted to evaluate two different ways of applying a use case model in an object-oriented design process resulting in a UML class diagram. |

# 2 State of the art

This section describes the state of the art and motivates the empirical studies undertaken as part of this thesis. The concepts of use case modelling and use case driven development are described, and software inspections and estimation methods are discussed briefly, because an inspection technique for use case models is proposed in one of the papers in the thesis (Paper II) and three of the papers investigate how use cases can be applied when estimating software development effort (Papers IV, V and VI).

## 2.1 Use Case Modelling

Developing or enhancing a software system requires a clear vision of the functionality that the system should provide. The development of a software system involves many different stakeholders, for example, representatives of the client, future users of the system, project manager and system testers in addition to the developers. The different stakeholders need to develop a common vision of the system, although they may have different views about it and, therefore, also different requirements for it.

The different stakeholders may also have different needs regarding the requirements specification of the system. The client and future user should, for example, be able to verify that they get the expected functionality. The project manager needs information that enables him or her to plan and estimate the project. The developers need the use case model to derive a design model for the system.

A use case model describes the functional requirements of a software system, that is, it provides a detailed description of what the system should do. Use case modelling is recommended as a technique for bridging the gap between descriptions that are meaningful to software users and descriptions that contain sufficient detail for modelling and constructing a software system. A use case model may therefore serve as a contract between the stakeholders of a system about its proposed behaviour.

### 2.1.1 Definitions

A *use case model* has two parts: the use case diagram and the use case descriptions. The diagram provides an overview of actors and use cases, and their interactions. An *actor* represents a role that the user can play when interacting with the system. A candidate actor is a human user or a hardware or software system that will use this system to achieve a goal. A *use case* represents an interaction between an actor and the system. A use case provides details of a requirement. A use case can be defined as follows:

*A use case is a set of stories of how actors will perform actions to achieve a goal under various conditions.*

An action is typically either a system change or a communication with the environment. As such, a use case describes one way in which a complex system can

be employed by its users. This means that a use case specifies the sequences of actions, called *scenarios*, enabled by the system, to achieve a particular goal. A use case will contain a scenario for the typical sequence of actions through the use case, and scenarios for alternative sequences of actions (Cockburn 2001). Use cases are primarily used to describe the outwardly visible requirements of a system, and each use case should describe something that will be valuable to one of the system stakeholders (Kulak & Guiney 2000).

Use cases can be described in many different ways, ranging from an informal, unstructured style to a more formal style approaching pseudocode (Hurlbut 1997). A common format is a template format that contains varying levels of detail depending on the actual project (Cockburn 2001).

Several recommendations and guidelines for how to apply the use case modelling technique have been proposed (Ben Achour *et al.* 1999, Regnell 1999, Armour & Miller 2000, Cox and Phalp 2000, Kulak & Guiney 2000, Cockburn 2001). These recommendations are mostly based on extensive experience of software projects. However, to the author's knowledge, only the guidelines constructed as part of the project CREWS (Ben Achour *et al*. 1999) and the modified variant, the CP guidelines (Cox and Phalp 2000, Cox and Phalp 2002) have been subject to empirical evaluation.

### 2.1.2 The use case modelling process

A system's use case model is often constructed in several steps (Kulak & Guiney 2000):

**Step 1**: Identify actors and their goals. Each unique goal is a candidate for a use case. For each use case, do the following:

**Step 2:** Detail the sequence of actions necessary to achieve the goal under normal conditions.
**Step 3:** Identify possible failure conditions for each action.
**Step 4:** Make an alternative sequence of actions for each failure condition.

A system's use case model may also be developed through several iterations, adding more details in each iteration (Kulak & Guiney 2000). In the first iteration, the use case model may be described at a high level without specifying details of each action. The use case model may supplement the client's requirements specification or be derived from it. In the second iteration a more detailed use case model is constructed, possibly together with domain experts or representatives of the client. Subsequently, more details may be added when the requirements change, become clearer or are needed for other reasons, for example regarding the user interface or the design of the system. Finally, the use case models may be reviewed by the developers only, or by several of the other stakeholders of the system.

The use case model describes the composite behaviour of a system, and can be used as input to many activities in a development project, such as design, testing and documentation. The quality of the use case model, therefore, is likely to have an important impact on the quality of the resulting software product.

Use case modelling is claimed to be a technique that facilitates the capturing of functional requirements because it stimulates discussion within a team about an upcoming system (Cockburn 2001). It is further claimed that use cases are easy to

understand also for stakeholders without specific training in the technique. The fact that customers and developers can achieve a common understanding of the requirements is often stressed as a primary motivation for applying the technique (Jacobson *et al.* 1992, Jacobson *et al.* 1999, Kulak and Guiney 2000).

A challenge when applying the technique, is to achieve an appropriate level of detail in the use cases. How does one include sufficient detail without ending up with an overwhelming and impractical expansion of the requirements? Too fine a granularity makes the use case model difficult to grasp, while too coarse a granularity hides the complexity (Kulak and Guiney 2000).

Guidance is therefore needed when eliciting and documenting functional requirements with the use case modelling technique. This is particularly important when developers are new to the technique.

The construction of a use case model is basically about how a team of people cooperates on obtaining a common understanding of a system through the construction of a model and text. Studies on program comprehension have used elements from cognitive psychology, and text comprehension in particular, to understand how software developers form different mental models of software programs (Burkhardt *et al.* 2002). Consequently, we may benefit from adapting elements from cognitive psychology also in the study of use case modelling to better understand how to construct and document use case models.

## 2.2 Use Case Driven Software Development

Use case driven software development means that use cases are applied as a primary artefact in many activities of a development project (Jacobson *et al*. 1992, Booch *et al.* 1999, Stevens and Pooley 2000), for example:

1. When planning the project, a use case model can be used when prioritising the functional requirements (Regnell 1999), and it can be used in project scheduling and in estimating development effort (Schneider and Winters 1998).
2. When designing an object-oriented system, a use case model can be used to create or validate a design model (Bennett *et al*. 1999, Jacobson *et al*. 1999, Arlow and Neustadt 2002).
3. When testing, a use case model can be used to derive test cases (Regnell 1999, Briand and Labiche 2001)
4. When documenting the system, it can be used as a basis for user documentation (Armour and Miller 2000).

### 2.2.1 Use case driven design

In a use case driven development process, a use case model, possibly in combination with a domain model, serves as the basis for deriving a design model with the classes necessary for implementing the system. One frequently recommended approach to deriving system classes from the use case model is to make sequence and/or collaboration diagrams for each use case scenario. The objects used in these diagrams, as well as those of a domain model, lead to the discovery of classes (Jacobson *et al.* 1999).

There are, however, different recommendations as to how the system classes should be derived from a use case model (Stevens and Pooley 2000, Liang 2003).

The choice of design process may have an effect on the resulting design model, which in turn may affect the development effort needed to both implement and maintain the system (Briand and Wüst 1999, Arisholm *et al*. 2001).

## 2.3 Inspections

Inspection techniques have proven valuable for detecting defects and improving quality of software documents from requirements documents to code (Fagan 1976, Doolan 1993, Gilb and Graham 1993, Briand *et al.* 1998b). There are inspection techniques for most documents produced in a software development project.

### 2.3.1 Definitions

An *inspection* is defined as a formal evaluation technique in which software requirements, design or code are examined in detail by a person or group, in order to detect defects, violations of development standards, and other problems (Cheng and Jeffrey 1996).

In *ad hoc* techniques the inspectors use a non-systematic way of identifying defects, while in *checklist-based* techniques the inspectors are provided with a list of general defect classes to check against (Cheng and Jeffrey 1996). The *scenario-based* technique, in which the inspectors are given specific instructions on how to search for defects (Shull *et al.* 2000), is another kind of inspection technique, which has given promising results.

A *review* is defined as a manual process that involves general defect classes against which to check document (Sommerville 2000). A *walkthrough* is a peer group review of a software document (Yourdon 1989).

### 2.3.2 State of practice

A recent survey on state of practice for reviews and inspections conducted among 226 employees in companies world-wide (primarily in Germany and Norway) (Ciolkowski, 2002) shows that inspections are more frequent than peer reviews and walkthroughs when it concerns requirements. The survey further shows that checklist-based techniques are the most common in inspections; they are used in 54% of the inspections, while *ad hoc* inspections are used in 31% of the cases, and scenario-based techniques are used in 10%. More than 50% of the respondents rated inspections as crucial for

- obtaining quality improvement,
- evaluating project status and
- enforcing project standards.

Very few found that inspections were unimportant in obtaining quality, but 56% of the respondents considered inspections to be very expensive.

The increasing use of UML has motivated the development of a family of reading techniques for UML diagrams (Shull et al. 1999), but no comprehensive inspection technique for use case models exists. The structuring of the functional requirements in

a use case model does, however, motivate the construction and use of an inspection technique that contains strategies for discovering defects adapted to this particular structure. The literature on use case models recommends reviews of use case models to ensure quality (Schneider and Winters 1998, Armour and Miller 2000, Kulak and Guiney 2000). Many organizations conduct such reviews with varying degree of formality. Several checklists for use case models have been proposed, but they are not part of a comprehensive inspection technique that, for example states who should participate in, and what should be the procedure of, an inspection of a use case model. Moreover, the checklists have been subject to no a empirical test.

## 2.4 Use Case Based Estimation

A large proportion of projects for the development of industrial systems significantly overrun budget, are delivered after schedule (or not delivered at all), or are not delivered with the specified functionality (PITAC 1999). Therefore, there is a need for research on how to improve estimation practices. *The estimation of software development effort* is concerned with predicting the effort that is necessary for implementing a particular software system and, possibly, determining a completion date.

Expert estimation appears to be the dominant strategy when estimating software development effort. A review of studies on estimation (Jørgensen forthcoming) shows that the accuracy of expert estimates can be improved as follows:

1. Combine estimates from expert with estimates based on other estimation strategies.
2. Estimate top-down and bottom-up independently.
3. Justify and criticize estimates.

This implies that estimation accuracy may be improved by supplementing expert estimates with estimates derived from a more formal estimation method. The function points method (Albrecht 1979) has been used successfully for many years, but is not adapted to an object oriented context.

A use case model defines the functional scope of the system to be developed. Attributes of a use case model may therefore serve as measures of the size and complexity of the functionality of a system. An estimation method that derives estimation parameters from a use case model, *the use case points method*, was introduced by Karner (Karner 1991). This method is influenced by the function points method.

Two other methods have been proposed for estimation based on use cases (Fetcke *et al*. 1998, Smith 1999). These methods respectively make assumptions about the relationship between use cases and function points, and between use cases and the number of lines of code (LOC) in the finished system. There is also a commercially available tool for use case based estimation (*Tassc:Estimator* 2002).

Many companies use a system's use case model informally in the estimation process, and some studies have found use case models to be suitable as a basis for estimation (Arnold and Pedross 1998, Martinsen and Groven 1998, Ribu 2001). However, little empirical evaluation of use case based estimation exists. We therefore have little knowledge of the conditions under which it is applicable, or of its accuracy compared with expert estimates.

# 3 Research Methods

Success in software engineering depends on a solid understanding of both the development process and the resulting products. It is therefore necessary to define process and product qualities, and evaluate successes and failures to learn from experience.

Empirical software engineering is the study of software engineering based on observations and experiences. The empirical research method has traditionally been used in social sciences and psychology, and is based on the proposal of a model followed by empirical validation. The main goal of empirical studies is to enable understanding and to identify relationships among different factors. The studies should be conducted and reported in such a way that practitioners, who are the audience for the research, are able to understand our theories and findings in the context of their work and values.

The overall aim of the field of empirical software engineering is to establish a scientific approach that can be used to predict how processes, methods and tools will perform in a given context. The scientific approach should comprise a set of research methods, theories and terminology. A theory for use case modelling would be useful to better predict how this technique will perform in a specific context and also for advancing the field in a systematic way.

To make software engineering more scientific, it is necessary to know how to use the available approaches to conducting empirical studies. The goals of this chapter are to:

- describe the current state of the art for research methods applied in empirical software engineering,
- discuss how to choose between and improve existing types of study, and
- show how different kinds of studies were applied in this thesis.

## 3.1 Phases of Empirical Studies in Software Engineering

Empirical research can be broken into four phases: The exploratory phase, the proposal phase, the analytical phase and the evaluative phase (Glass 1994). A simplistic view of the empirical research process in software engineering is shown in Fig. 1, which is based on (Wallace 1969, Næss 1980). The research process can be approached in two different ways.

- Observations or exploratory studies are the most common starting point (underlined in Fig. 1) in software engineering. Information is gathered or aggregated, and the research community becomes familiar with the basic facts, setting and concerns of a phenomenon.
- Alternatively, information may be deduced from antecedent knowledge, perhaps in the form of theories from other fields of study.

Both approaches can be used to formulate hypotheses. In the proposal phase, a set of hypotheses that will comprise a theory, a model or a framework, is formulated.

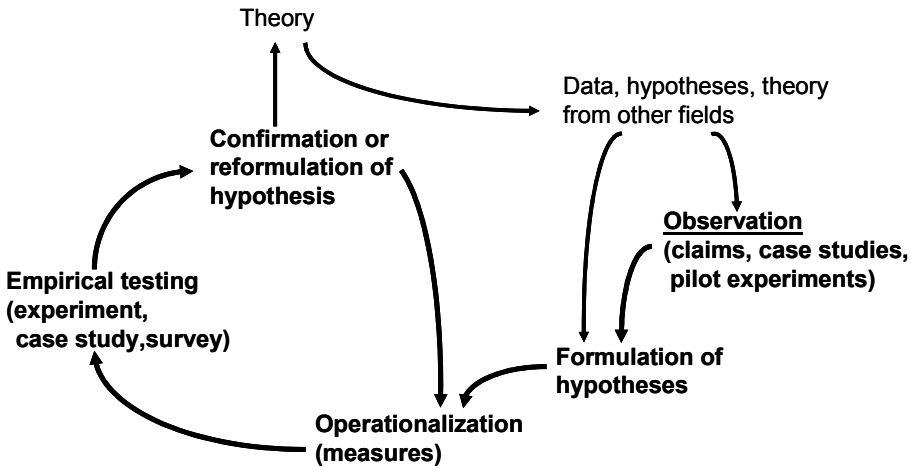


**Fig 1. The empirical research process in software engineering**

To conduct studies in which the hypotheses can be tested and valid results produced, there is often a need for further exploratory studies. First, to determine the feasibility of conducting the actual research, and second, to develop and improve a study design, including the development of techniques for measuring and locating future data. Hence, in the analytical phase, the hypotheses are rendered operational and corresponding measures are developed.

The hypotheses are subjected to empirical testing in the evaluative phase. The results may lead to the acceptance of a theory, model or framework, pending further testing or to the reformulation of the hypotheses.

The research process in empirical science is typically cyclic. Since it is not possible to establish the truth of a theory definitively, but only to find confirming or fail to find falsifying instances, theories will always be open to the possibility of refutation by future observations or tests. After a period of testing, if no falsifying instances have been found, the theory may be regarded as having been accepted as workable by the scientific community. At present, however, empirical software engineering research is typically in its theory-creating stage, which means that studies are concerned with formulating, operationalizing and testing hypotheses.

### 3.2 Research Methods in Software Engineering

The empirical method is one among others that are applied in software engineering and which together may be said to form at least part of the scientific method as a whole. Two other methods are the analytical method and the engineering method (Adrion 1993):

1. The analytical method proposes a set of axioms, from which a theory is derived. Results from the theory are deduced, and if possible, compared with empirical observations. The analytical method requires more formalism than the empirical method when deducing consequences from a theory.
2. The engineering method enables existing solutions to be improved upon. Based on the results of testing existing solutions, better solutions are suggested, which in turn are developed, measured and analyzed. The engineering method requires less formal evaluation, for example no hypothesis testing, than the empirical method.

Three kinds of studies tend to dominate research in empirical software engineering; controlled experiments, case studies and surveys. The following sections describe the features and applicability of these three approaches.

### 3.3 Controlled Experiments

A controlled experiment isolates, in an artificial way, the phenomena to be investigated. It can be defined in the following way:

*A controlled empirical investigation into some phenomenon with a clearly stated hypothesis and random allocation of subjects to different treatments.*

(Delgiannis *et al*. 2002)

A controlled experiment is characterized by the following (Wood *et al*.1999):

- an explicit experimental design
- one or more treatments to two or more groups
- random assignment of subjects
- the effect of the treatment is measured
- direct comparison amongst groups is performed

To control experimental conditions carefully, experiments are mostly conducted in artificial surroundings, typically with students as subjects in a laboratory setting. A *field experiment* in software engineering is conducted within an organization, but the participants perform tasks other than those they usually perform, or perform their usual tasks in a way differently from that in which they normally perform them.

The assumptions underlying an experiment are that it is possible to predict output based on input, that albeit it is possible to capture the relevant context variables to control them and that a problem can be reduced to the control of a predetermined set of variables. A set of recommendations on how to conduct a controlled experiment is given in (Pfleeger 1995):

1. Define the goal of the experiment and ensure that a controlled experiment is the most appropriate kind of study.
2. Formulate hypotheses and design situations that is, subjects, tasks and development environments, that are capable of testing the hypotheses or implications of the hypotheses.
3. Develop experimental materials and prepare subjects.
4. Apply the treatments to the experimental subjects in accordance with the experimental design.
5. Analyze collected data using statistical techniques.
6. Conclude the results, that is, discuss the hypotheses in the light of previous knowledge and the results from the experiment, and document all the key aspects of the research.

Experiments can be used in exploratory studies, but are most often used in theory-testing studies, preceded by an exploratory phase consisting of case studies, surveys or pilot experiments that form the basis for the hypotheses. Experiments are typically used to evaluate relationships between phenomena that have been observed in case studies.

### 3.3.1 Strengths and weaknesses of controlled experiments

The strengths of controlled experiments are the high degree of control that it is possible to maintain over the phenomena to be studied, and the possibility of designing them so that their results are statistically valid. The major weakness is that it can be problematic to generalise the results beyond the experimental setting, due to its artificial nature. The goal of research in empirical software engineering is the transfer of useful results to industry. To convince industry about the external validity and applicability of the experimental results, a strong resemblance between the experimental setting and actual practice in the work environment is important.

One of the largest challenges, is perhaps, to recruit a sufficient number of realistic subjects, that is, subjects that are drawn from the actual population about which we wish to make (Kitchenham *et al.* 2002). It may also be difficult, and involve high costs, to maintain the cooperation of the subjects for a sufficient amount of time to enable them to perform tasks of a realistic size. Software development mostly involves the use of development tools, but the use of professional software development tools in an experiment poses additional challenges (Sjøberg *et al.* 2003). Even if the development environment is realistic, the experimental setting may contribute to an unrealistic experimental context, which in turn makes it difficult to transfer results from experiment to industry.

## 3.4 Case Studies

Case studies are the most common kind of study carried out in cooperation with industry in empirical software engineering research. The researcher is more of an observer than is the case in a controlled experiment (Wohlin *et al*. 2000). A case study allows in-depth understanding of one particular case or development project; that is, of one particular phenomenon or a few related phenomena.

The understanding of a phenomenon requires the capturing of sufficient details about it and the context in which it occurs. Data is collected for specific purposes throughout the study, and one or several different techniques for data collection may be applied, e.g., interviews, questionnaires, archival information, surveillance of a program and/or video and voice recording.

The following guidelines for conducting software engineering case studies, among others, are suggested in (Arisholm *et al*. 1999):

1. Know the company and ensure the necessary backing from it.
2. Be explicit about research goals and company goals.
3. Present interim results frequently.
4. Popularize the results of the study in addition to publish scientific papers.

Case studies are common as explorative studies, or to obtain a rich description of a phenomenon and the context in which it occurs. However, the most important application of case studies is to explain the causal links in situations that are too complex for survey or experimental approaches (Dybå 2001). When the goal is to construct a theory, *typical* projects are studied whereas when the aim is to test theories, *special* situations that could *falsify* the hypotheses should be sought.

### 3.4.1 Types of case study

There are different variants of case studies, depending on the degree of control and desired change of working practices:

- A case study in which there is (almost) no intervention from the researcher can be classified as an observational study.
- A case study characterized by planned and deliberate changes to the organization under study can be classified as action research.
- Several case studies may form a multi-case study, which provides a broader picture from which both researchers and industry can draw knowledge.

### 3.4.2 Strengths and weaknesses of case studies

The major strength of a case study is the possibility of studying a phenomenon in depth in a realistic context. Weaknesses are related to lack of control, which leads to problems with generalizing the results. Moreover, the data collected may be interpreted in different ways, and the intervention of the researcher may affect the organization studied. Therefore, it may be difficult to analyze causes of the observations made in the study.

Several practical challenges face the researcher when undertaking a case study. For example, it may be difficult for the organization to find time to participate, and they may be unwilling to give the researcher access to all their projects. This may lead to a bias in the selection of projects studied. The organization may expect quick and easily applicable results, which may run counter to the goals and practice of the research. Therefore, it may be difficult to persuade a company to sustain a software process initiative (Arisholm *et al*. 1999).

## 3.5 Surveys

Surveys can be considered as research-in-the-large, because they typically try to capture what is happening over a large group of projects, and surveys allow the capture of many variables (Kitchenham 1995). In a survey, data is collected in a standardized way for a particular population. If the survey is conducted on a large scale, the results may generalize to many organizations and projects. Survey research has three distinct characteristics:

- It produces quantitative descriptions of a number of aspects of a study population.
- Information is collected by asking people structured, predefined questions.
- Information is collected from a fraction of the study population, and collected in such a way that the findings can be generalized.

A survey may be used in the construction of a theory if the questions are open-ended. However, in practice, they are most often structured and applied in the testing of theories. Surveys are often performed in retrospect, to draw explanatory conclusions. There are different techniques for conducting surveys, for example, literature survey, use of questionnaires and interviews.

### 3.5.1 Strengths and weaknesses of surveys

The strengths of surveys are realism and that it is practical to replicate the surveys. A weakness is that it is difficult to interpret the results, because there may be many ways of interpreting answers given by unfamiliar subjects. Also, the experimental unit is often a problem, because questions are often given to individuals, but concern an organization.

It is often difficult to obtain a random sample from the target population, that is, the population that we wish to draw conclusions about. Therefore, *convenience samples* are often used instead (Cunningham 1997). The phrasing of the questions and obtaining a high enough response rate are other challenges involved in conducting surveys (Hufnagel and Conca 1994).

## 3.6 Obtaining Results that can be Generalized

The three types of study described above can be used to investigate the same topics, but with focus on different aspects. The choice of research method will be guided by:

- the purpose of the research which may be to explore, describe or explain a phenomenon,
- the type of research,
- the phase of the research (described in Section 3.1), and
- the preferences of the researcher.

In all of the three types of study described above, it may be difficult to obtain results that can be generalized outside the study setting. In survey research, the results may be generalized if the respondents are representative of the population about which we wish to make claims. In case studies, a rich description of the context of the study is

provided, with the aim of generalising to similar settings. In controlled experiments, the reduction of the topic under study and the control of the involved variables are aids to generalising the results.

"Even though our approaches to empirical software engineering research illuminate the relationships among variables, they may in fact limit what we see" (Pfleeger 1999). The approaches apply limited measures and assume rationality, but because of the large human element, perfect rationality with respect to the phenomena being studied cannot be expected. Hence, it is debatable whether it is feasible to obtain universal and predicting theories by means of reduction on the topics of software engineering. Context-dependent knowledge may thus be more valuable than the search for universal and predicting theories. Capturing the context of the phenomenon under study in sufficient details, it may be possible to generalize on the basis of one case and thereby contribute to scientific progress (Flyvebjerg 1991). Consequently, the capture of sufficient context is important in both case studies and experiments if the goal is to obtain results that are applicable outside the study setting.

Our aim should be to reduce uncertainty, if it is infeasible to determine cause and effect. We therefore need to devise strategies to help us deal with the imperfect knowledge and uncertainty in our measures and models. Such strategies could include stochastic models that give us the likelihood that one option is better than another under certain conditions (Pfleeger 1999).

With respect to capturing sufficient context, the participants' background may be among the most important context variables. This is discussed in further detail in Section 3.7. Two orthogonal strategies can be used to obtain sufficient contextual information to predict which option is better than another in a specific context: combining several studies or increasing the realism of controlled experiments. These strategies are discussed in Section 3.8 and 3.9.

### 3.7 Background Information about Participants

Papers describing empirical software engineering studies often do not characterise the participants' competence, experience and educational background, and the authors seldom justify the extent to which their subjects are representative of the software engineers who usually perform such tasks. This leads to several problems:

- The results may not be trustworthy, that is, the participants may not be representative of the target population. The sample recruited may be biased in some way, for example, a company may only be willing to let the software engineers who are least experienced or least in demand take part in an experiment.
- Comparing the results from an original study with a replicated study is difficult.
- Successful transfer of the results into industrial practice is less likely.
- To generalise from experiments with a given group of subjects, it is necessary to possess information about the ability and the variations present in members of the target population (Basili *et al*. 1999). Depending on the topic being studied, it would be relevant to know the variations regarding competence, productivity, education, experience (including domains), age, etc. (Some of this information may be highly controversial.) A challenge for the measurement of these attributes is to

define good measures that can be used in practice. For example, how can competence and productivity be measured? In practice, there is a need for meaningful substitute measures for the attributes that cannot be measured directly.

Such background information can be used in several ways, for example, to determine

- the target population for which the results are valid, and
- the extent to which the results of the treatments depend on the collected background information, e.g., it might be that certain design principles are easier to understand for experienced professionals than for novices.

It would also be interesting to identify the variations within the same company versus between companies, variations between in-house professionals versus consultants, etc. For example, in-house software development may differ from development projects run by consultancy companies. Nevertheless, knowledge about the effect of a certain technology among consultants or even students may still be useful when knowledge of the effect of the technology in a company's own environment is lacking.

## 3.8 Combining Studies

Research in software engineering has so far been characterized by the conduct of individual studies, while the aim should be to combine studies to develop a better understanding of the software development process (Basili *et al*. 1999, Shull *et al.* 2002). There is an increased commitment to the replication of studies in a variety of environments. The large number of context variables requires replications in which the context variables are systematically captured and varied. Also, it is often difficult to satisfy all criteria for validity in just one study. A framework for a family of experiments (Basili *et al*. 1999), and experimental packages to facilitate the replication of studies on reading techniques have been proposed (Shull *et al*. 2002).

However, different research methods allow us to study different aspects of a topic. Hence, different research methods should be combined. For example, relations that we believe to have observed in case studies or surveys may receive confirmation or be refuted through controlled experiments, and vice versa. The combination of research methods also requires the thorough collection of context variables, to ensure comparability between the studies.

In order to approach use case modelling scientifically, a large number of studies are needed to produce sufficient empirical evidence to allow the construction and test of a theory. Generally, it is also difficult to amalgamate study results in software engineering, because technology changes quickly. Research may be rendered more efficient by the use of experimental packages for replications of studies (Shull *et al.* 2002) and the development of tool support for conducting studies, for example, the tool described in (Arisholm *et al.* 2002).

## 3.9 Increasing the Realism of Experiments

In case studies, the lack of control of variables entails a lack of explanatory power. In controlled experiments, the lack of realism is an important obstacle to the transfer of research results to industrial applications. The context of a typical experiment in software engineering is remote from the actual work practice of software engineers (Sjøberg *et al.* 2002).

It is difficult to ensure control in case studies, at least from the position of an outsider, such as a researcher. Moreover, it is not always possible to evaluate new methods, techniques or tools through case studies. The technology to be evaluated may not yet be in widespread use, or there may be other difficulties related to performing case studies, as described previously. It is, however, possible to increase the realism of experiments by using more realistic tasks, recruiting suitable subjects and conducting the experiment in a more realistic environment. Nevertheless, the research results must, at least partially, be obtained in an industrial context if they are to be of interest to industry.

### 3.9.1 Realistic tasks

Regarding size, complexity and duration of the involved tasks, most experiments in software engineering seem simplified and short-term, in that "the experimental variable must yield an observable effect in a matter of hours rather than six months or a year" (Harrison 2000). Such experiments are hardly realistic, given the tasks of building and maintaining real, industrial software, particularly since many of the factors we wish to study require significant time before we can obtain meaningful results.

Increasing the duration of the studied tasks will therefore produce results that are more relevant to software practitioners because of a closer resemblance to actual work practice. Varying the duration will also provide the opportunity to study the effects of the factors we wish to study in different contexts.

### 3.9.2 Realistic subjects

One should be careful about the extent to which the selected subjects in an experiment represent the target population. Students are most often used as subjects, and in some cases they may be representative of the actual population (Høst *et al* 2000), but in other situations they probably are not, depending on the research question. A frequent criticism of empirical studies is that the subjects were students, and that the results consequently cannot be generalized to professional software developers. Students may, however, be very useful as subjects in exploratory studies in which they can be used to establish a trend and give some indications about how a method performs. Some hypotheses can be eliminated, and most importantly, students can be used to test and debug experimental designs (Tichy 2000).

Students, or junior professionals, may require more guidance in the form of methods or tools than do professionals with long experience, and may also be more likely to adapt new techniques. Hence, it may in many cases be relevant to study the

effects of particular methods or tools on students' performance, but the extent to which methodology taught in university courses is actually adopted by the student after they have graduated calls for further studies (Kautz and Pries-Heje 1999).

The results from experiments with students as subjects are, however, often of little interest to industry.

### 3.9.3 Realistic environments

Even when realistic subjects perform realistic tasks, the tasks may be carried out in an unrealistic manner. The challenge is to configure the experimental environment with an infrastructure of supporting technology (process, methods, tools, etc.) that resembles an industrial development environment. Traditional pen-and-paper-based exercises used in a classroom setting are hardly realistic for dealing with relevant problems of the size and complexity of most contemporary systems. Artificial experimental settings can cause many threats to validity.

To test the effect of using a professional modelling tool (Tau UML Suite from Telelogic), in one of the studies conducted as part of this thesis (Paper VII), half of the subjects used pen and paper, while the other half used the modelling tool. The results of the two groups were different.

## 3.10 Overview of Studies and Research Methods Applied

Because few studies have been conducted on how to construct and apply use case models, the studies reported in this thesis are exploratory. The aim was to develop an understanding, based on observation and analysis, that could form the basis for more rigorous studies with the long-term aim of developing a theory for the construction and application of use case models in software development projects.

Table 2 gives an overview of the reported studies, showing the main purpose of each study, the research method applied, how the subjects were recruited and how the data was collected. Sections 3.3–3.5 discuss strengths and weaknesses of the different types of study, some of them are exemplified in the columns "Advantages of design" and "Disadvantages of design". The content of these two columns also indicate the validity of the results of each study.

**Table 2. Overview of studies**

| Paper | Main purpose of study | Research method | Subjects | Data collection | Advantages of design | Disadvantages of design |
|---|---|---|---|---|---|---|
| **I** | Investigate how different guidelines for use case modelling affect the quality, in particular, the understandability, of use case models. | Experiment. Conducted as part of student project. Teams of subjects used different guidelines. | 139 students in 31 teams. | The use case models were evaluated. The subjects answered questionnaires about the functionality of the use case models. | Realistic size of task because the use case modelling process took approx. 2 weeks. Large amount of groups of students, which led to statistical significance. | Students, with little experience with use case modelling as subjects. |
| **II** | Propose and evaluate an inspection technique for use case models. | Case studies on student project and contr. Exp. with students as subjects. | 157 students in 58 teams in case studies, 46 students in exp. | Lists of defects detected and effort spent collected from both case studies and experiment. | Combination of different research methods. | The same subjects in both studies may have affected the results. |
| **IV** | Evaluate a method for estimation based on use case models. | Case studies on three software development projects. | Software development company. | Use case model and project data from one "live" project and two historical projects. | Realistic context because the estimation method was evaluated on real software development projects. | Some insecurity about actual effort on projects complicates evaluation. |
| **V** | Compare expert estimates from teams of developers with an estimate produced using a method based on use cases. | Observational study on teams of professional software developers. Part of courses on use case modelling. | 37 professional developers in 11 teams. | Information about the teams and their estimates was collected. | Realistic subjects as they were very experienced software developers with reasonably good knowledge of the project to be estimated. | The context of the study may have been different from situations in which the subjects usually do the estimation. |
| **VI** | Investigate how use case based estimation can be tailored to a company. | Survey in the form of interviews. | 11 project managers and senior developers. | Semi-structured interviews were taped and transcribed. | In-depth interviews with experienced software developers. | Convenience sample of interviewees. |
| **VII** | Compare two alternative ways of applying a use case model in object-oriented design. | Controlled experiment with students as subjects. | 59 students as subjects. | The design models were evaluated, time spent and comments written by the participants were collected using SESE | Realistic development environment with use of professional UML modelling tool. | Students, with little experience with UML as subjects. Small task. |

# 4 Results

The questions that motivated the research presented in this thesis were described in Section 1.1. In terms of construction, documentation and validation of use case models, the use of both guidelines and an inspection technique was investigated. The results from these studies support existing claims that guidelines based on templates produce use cases of better quality than do those that do not specify how to document individual use cases. The results further show that an inspection technique tailored to the specific format of use case models may be useful for detecting defects them. The results also give advice on how such an inspection technique can be further improved.

With respect to the application of use case models, the estimation of development effort and the design of class models were studied. The results are promising with respect to the possibility of applying use case models to improve the accuracy of effort estimates. The results of the studies on use case based estimation also provide recommendations for how the technique can be applied in software development companies. Moreover, the results show the trade-offs of two different ways of applying a use case model in an object-oriented design process.

Too few empirical studies have been conducted on the topic of use case models, including those reported in this thesis, to provide complete answers to the research questions proposed in Section 1.1, but the studies of this thesis can represent an incremental step answering them. Section 5 discusses how the research could proceed towards a theory for the construction and application of use case models. The findings in the respective papers are summarized below.

## 4.1 Summary of Individual Papers

This section summarizes the individual papers in this thesis. Most of the papers are written in cooperation with other authors, but the core of the work, that is, the main ideas, the designs of the studies and the majority of the writing was done by the author of this thesis. In all the papers, the authors are listed according to contribution, not alphabetically.

### Paper I: Quality and Understandability of Use Case Models

*Bente Anda, Dag I.K. Sjøberg and Magne Jørgensen*

J. Lindskov Knudsen (Ed.): ECOOP 2001 – 15th European Conference on Object-Oriented Programming, Budapest, Hungary, June 18-22, 2001, pp. 402-428, LNCS 2072, Springer-Verlag.

Use case models are used in communication among stakeholders in software development projects. It is therefore important that the use case models are constructed in such a way that they support the development process and promote a solid understanding of the requirements among the stakeholders. Nevertheless, there

is no commonly agreed set of quality attributes for use case models, and there are few empirically validated guidelines on how to construct them.

This paper proposes a set of quality attributes for use case models. Three common sets of guidelines are highlighted and contrasted with respect to quality and understandability, in particular.

The paper describes an exploratory study, in which the three different sets of guidelines were used for constructing and documenting use case models. An experiment with 139 undergraduate students divided into 31 groups was conducted. Each group used one out of the three sets of guidelines when constructing a use case model from an informal requirements specification. After completing the use case model, each student completed a questionnaire.

The results of the experiment indicate that guidelines based on templates support the construction of use case models that are *easier to understand* for the readers, than do guidelines without specific details on how to document each use case. The guidelines based on templates were also considered to be the most useful when *constructing* use cases. In addition to better understandability, the experiment indicates that the guidelines based on templates result in better use case models with respect to other quality attributes, such as the correct identification of actors and use cases and a balanced level of detail in the descriptions of the flow of events of the use cases. The results further indicate that it may be beneficial to combine the template guidelines with another set of guidelines that focus on the *documentation* of the flow of events of each use case.

### Paper II: Towards an Inspection Technique for Use Case Models

*Bente Anda and Dag I.K. Sjøberg*

Proceedings of SEKE'02 – 14th IEEE Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 15-19, 2002, pp. 127-134.

Software inspection is regarded as one of the most efficient methods for verifying software documents. There are inspection techniques for most documents produced in a software development project, but no comprehensive inspection technique exists for use case models.

This paper presents a taxonomy of typical defects in use case models and proposes a checklist-based inspection technique for detecting such defects. The inspection technique was evaluated in a case study of 27 student projects and a controlled experiment with 45 undergraduate students as subjects. The results indicate that inspections are useful for detecting defects in use case models and provide suggestions for how to improve the proposed inspection technique. The results also indicate what types of defect are discovered by different stakeholders in a software development project.

### Paper III: Understanding Use Case Models

Bente Anda and Magne Jørgensen

Proceedings of Beg, Borrow, or Steal Workshop, International Conference on Software Engineering, Limerick, Ireland, June 5, 2000, pp. 94-102.

The use case model can serve as a means of communication among the different stakeholders in a project. With a view to reducing the possibility of misunderstandings and different interpretations, this paper investigates how different stakeholders understand and interpret use case models. Low comprehension or differences in interpretation may indicate that more attention should be given to specifying the requirements. If this is not feasible, it may be necessary to assume a higher risk when planning and estimating the project.

This paper discusses how results from research in cognitive psychology on how humans understand text, in particular, schema theory, may be used to (a) improve our understanding of how use cases are read and understood by different stakeholders in a project, (b) develop methods for measuring comprehension of use case models and (c) analyse differences in interpretation of use case models.

### Paper IV: Estimating Software Development Effort based on Use Cases – Experiences from Industry.

*Bente Anda, Hege Dreiem, Dag I.K. Sjøberg and Magne Jørgensen*

M. Gogolla and C. Kobryn (Eds.): UML 2001 – 4th International Conference on the Unified Modeling Language, Toronto, Canada, October 1-5, 2001, pp. 487-502, LNCS 2185, Springer-Verlag.

This paper reports the results of three industrial case studies on a method for software development effort estimation based on *use case points*. The results support existing claims that use cases can be used successfully in estimating software development effort. The results further indicate that the guidance provided by the use case points method can support expert knowledge in the estimation process. The paper also points out a number of issues that may be relevant for other organizations that want to improve their estimation process applying use cases, in particular, the fact that the design of the use case models has a strong impact on the estimates.

### Paper V: Comparing Effort Estimates Based on Use Case Points with Expert Estimates

*Bente Anda*

Proceedings of EASE 2002 – Empirical Assessment in Software Engineering, Keele, UK, April 8-10, 2002.

This paper reports the results from a study conducted to evaluate the *use case points method*, by comparing it with expert estimates. The use case points method gave an estimate that was closer to the actual effort spent on implementing the system than most estimates made by 37 experienced professional software developers divided into 11 groups (MRE of 0.21 versus MMRE of 0.37).

These results show that the combination of expert estimates and method-based estimates may be particularly beneficial when the estimators lack specific experience with the application domain and the technology to be used.

### Paper VI: Improving Estimation Practices by Applying Use Case Models

*Bente Anda, Endre Angelvik and Kirsten Ribu*

M. Oivo and S. Komi-Sirviö (Eds): PROFES 2002 – 4th International Conference on Product Focused Software Process Improvement, Rovaniemi, Finland, December 9-11, 2002, pp. 383-397, LNCS 2559, Springer-Verlag.

This paper proposes a tailored and potentially improved version of the use case points method and suggests how estimation practices can be improved by applying it, based on interviews with 11 project managers and senior developers of a software development company. The paper further discusses prerequisites for successful use of the use case model and the use case modelling process for the development process within the company.

### Paper VII: Applying Use Cases to Design versus Validate Class Diagrams – A Controlled Experiment Using a Professional Modelling Tool

*Bente Anda and Dag I.K. Sjøberg*

Simula Research Laboratory Technical Report No. 2003-01, 2003.

This paper describes alternative ways of applying a use case model in an object oriented design process, that results in a UML class diagram. Two alternative processes were evaluated in a controlled experiment with 53 students as subjects. One process was use case driven, while the other was a responsibility-driven process in which the use case model was applied as a means to *validate* the resulting class diagram. Half of the subjects used the modelling tool Tau UML Suite from Telelogic; the other half used pen and paper. The results show that the validation process led to class diagrams that implemented more of the requirements. The use case driven process did, however, result in the class diagrams being structured better.

# 5 Future work

Several areas of further research have been identified in the papers, and some additional directions for future work were pointed out in Section 3. A number of suggestions for future work are summarised below.

## 5.1. Improving the Techniques Investigated in this Thesis

This thesis suggested how guidelines could be employed in the use case modelling process, proposed an inspection technique for use case models has been proposed and evaluated an existing estimation technique for use case models. All these techniques need to be further improved and refined.

### 5.1.1 Inspections of use case models

The taxonomy and the inspection technique proposed in Paper II need to be improved through further studies in order to become generally applicable. In particular, it would be useful to investigate in greater detail what kind of information in the use case model is important for the various stakeholders in a software development project. The inspection technique should be tailored to the needs of the different stakeholders.

### 5.1.2 Use case based estimation

Use case based estimation seems to be promising as a method for supporting expert estimates of software development effort. However, many issues remain to be investigated before the technique can be expected to obtain widespread use.

- Software development often consists in changing an existing system. This entails that the method for use case based estimation should be enhanced to support the measuring of the size of a change to the use case model.
- The complete functionality of a (part of a) software system will frequently not be specified at the start of a software development project. An improvement to the method for use case based estimation would therefore be to include the possibility of specifying a particular amount of functionality to be developed without being specific about its content.
- The use case points method prescribes how to measure the size and complexity of a use case model. In certain cases this way of measuring size and complexity is inappropriate. Hence, Paper VI suggests an alternative approach to this measure. A topic for future work is to develop a more comprehensive set of metrics for measuring the size and complexity of use cases that will cover a wider range of formats.
- The success of use case based estimation seems closely related to the ability to assess the number and size of the use cases for the system. It would therefore be useful to investigate the kind of knowledge and experience that is required for such assessment.
- Expert estimation is the most frequently used strategy for estimating software development effort. Future work should therefore focus on how the technique can be combined with expert estimates; that is, identifying the respective strengths and weaknesses of expert estimates and use case based estimates.

### 5.1.3 Combining different techniques

This thesis proposed and studied both construction guidelines and an inspection technique as means of ensuring the quality of use case models. Moreover, it investigated the application of use case models to the estimation of development effort and to the design of object-oriented software systems. The combination of the results of this research would therefore represent interesting topics for future work, for example:

- comparing the efficiency of guidelines with that of inspections; that is, investigating which of these approaches is the most effective in a specific context, and also how the two techniques can be combined,
- investigating the impact of the quality of the use case model on the accuracy of the project estimates based on the use case model and
- investigating the impact of the quality of the use case model on the ability to identify correctly the necessary elements and the structure of a design model.

## 5.2 Investigating other Aspects of Use Case Modelling

Other aspects of use case modelling that are vital for the general applicability of the technique have not yet been subject to much research. Some examples are described below.

### 5.2.1 Evolution of use case models

To get full value from the work spent on the construction of a use case model, it should be applied extensively in the development and subsequent enhancements of the system. However, use cases are often perceived as belonging to and driving the initial phases of the development project, but they are rarely updated after the analysis phase of the project is completed, and very rarely after the system has become operational. Therefore, methods and tools for supporting the change and evolution of use case models in the development lifecycle are required (Weidenhaupt *et al.* 1998).

### 5.2.2 Handling large use case models

Many software systems are large and complex. This will also be true for the corresponding use case models and individual use cases describing their functionality. Consequently, there is a need for methods and tools for structuring and handling large use case models.

### 5.2.3 Reusing use case models

Reuse is promoted as a way of producing more efficient software, both in the design and coding phases. In practice, the requirements of a system will often be based on an existing system. Although little methodology exists for the reuse of use case models (Biddle *et al*. 2002), the concept of use case patterns has been proposed (Liwu 2002) and should be explored further.

## 5.3 Investigating the Proposed Techniques under Various Conditions

Investigating techniques, methods and tools in various contexts is important for the evaluation of their general applicability. For example, generalizing the use of the technology from novices to experienced professionals, from individuals to teams, from small to large projects, from one application domain to another, etc., calls for further empirical evaluation in which the effect of different contextual factors are investigated. This also applies to the techniques and methods proposed and evaluated in this thesis. It might be valuable to investigate how the use case modelling technique can be tailored to the needs of different users.

### 5.3.1 The needs of different users

In one of the studies of this thesis (Paper VI), professional developers were asked what they considered to be the main challenges when applying use case modelling. It would probably be useful to perform a larger survey among developers with varying levels of expertise on what they consider to be the main difficulties of use case modelling.

### 5.3.2 Effect of background knowledge

An interesting topic for future study would be to supplement the survey suggested above with an investigation of the prerequisites for successfully constructing and applying use case models in terms of proficiency with the technique. For example, what kind of and how much training and experience are needed? Studies from object-oriented design show that the difficulties associated with applying a technique varies with varying levels of experience of the users of the technique (Sheetz 2002). Moreover, the knowledge of the application domain and similar systems may also be important. One way of investigating these issues would be to conduct an experiment in which several teams of developers with varying backgrounds construct a use case model for a given system, and then subsequently develop the system according to a use case driven development process.

## 5.4 Investigating Theories from other Fields of Study

Software engineering is an immature discipline. The software engineering community should seek to identify elements of theories from established sciences that could help to make software engineering a more mature scientific discipline. This requires the identification of the basic problems of software engineering. In the specific area of use case modelling, it may be beneficial to investigate those elements from cognitive psychology that are related to the construction and understanding of text and diagrams. One approach to investigating this issue is presented in Paper III in this thesis.

## 6 Concluding Remarks

Requirements handling is vital for the success of software engineering. Regarding functional requirements, use case modelling has become a widespread and promising technique. Despite the apparent simplicity of this technique, it is not straightforward to apply it in specific software development projects. Organizations that apply the technique need advice on how to adapt it to the local environment. This motivates the need for empirical studies on how the technique can be successfully applied under various conditions.

This thesis investigated the use case modelling process, and the role of the resulting use cases in other activities of a software development project, through a number of empirical studies, both experiments and case studies. These studies proved difficult to undertake, particularly because they involved human subjects. In addition to increased insight into some aspects of the use case modelling technique, this work provides a basis for further empirical studies on this technique. These studies may represent a step towards a theory for use case modelling.

The studies conducted as part of this thesis were exploratory because only a few empirical studies on use case modelling had been conducted previously. The motivation for this work was to investigate empirically use case modelling from different angles, rather than going deeply into one aspect. The research required the development of original experimental material, including a more precise definition of the concept of quality of use case models. The field of use case modelling contains a large amount of concepts and terminology that are not precisely defined.

In retrospect, more effort in this thesis could have been put into rendering clear and precise the terminology related to use case modelling. The experimental designs could also have been more elaborate, with respect to both choice of treatment and regarding the aspects that were measured.

Nevertheless, my priority in this thesis was to conduct empirical studies that both help to clarify important concepts in the field and form a basis for further empirical studies. Such studies will make use case modelling more valuable and make it become more widespread, which in turn may improve object-oriented development in general.

*PAPER I:*

# Quality and Understandability of Use Case Models

*Bente Anda, Dag Sjøberg and Magne Jørgensen*

J. Lindskov Knudsen (Ed.): ECOOP 2001 - 15th European Conference on Object-Oriented Programming, Budapest, Hungary, June 18-22, 2001, pp. 402-428, LNCS 2072, Springer-Verlag.

## Abstract

Use case models are used in object-oriented analysis for capturing and describing the functional requirements of a system. Use case models are also used in communication between stakeholders in development projects. It is therefore important that the use case models are constructed in such a way that they support the development process and promote a good understanding of the requirements among the stakeholders. Despite this, there are few guidelines on how to construct use case models.

This paper describes an explorative study where three different sets of guidelines were used for constructing and documenting use case models. An experiment with 139 undergraduate students divided into 31 groups was conducted. Each group used one out of the three sets of guidelines when constructing a use case model from an informal requirements specification. After completing the use case model, each student answered a questionnaire.

The results of the experiment indicate that guidelines based on templates support the construction of use case models that are easier to understand for the readers, than guidelines without specific details on how to document each use case. The guidelines based on templates were also considered as the most useful when constructing use cases. In addition to better understandability, our experiment indicates that the guidelines based on templates result in better use case models regarding also other quality attributes. Our results further indicate that it may be beneficial to combine the template guidelines with another set of guidelines that focus on the documentation of the flow of events of each use case.

**Keywords.** Object-oriented analysis, Requirements specification, Use Cases, UML, Understandability, Experiment

# 1 Introduction

The concept of use case modelling was introduced by Jacobson in 1992 [1]. He also introduced a use case driven approach to software development. Since then, use case modelling has become a popular and widely used technique for capturing and describing the functional requirements of a software system. It is also used as a technique for bridging the gap between descriptions that are meaningful to software users and descriptions that contain sufficient details for modelling and constructing a software system. A use case model has two parts, the use case diagram and the use case descriptions. The diagram provides an overview of actors and use cases, and their interactions. The use cases' text details the requirements. An actor represents a role that the user can play with regard to the system, and a use case represents an interaction between an actor and the system.

In a use case driven software development process, the use case model is recommended as a primary artefact and is, for example, input to design and a basis for planning, estimation, testing and documentation. In addition, a use case model will often be part of a contract between the development organization and the customer regarding the functional requirements of the system to be developed. Therefore, the quality of the use case model may have a large impact on the quality of the resulting software system. Nevertheless, there is no commonly agreed theory on how to construct use cases, and there are different opinions about what constitutes quality in use case models.

UML adopts use cases but offers little advice on how to apply them. The support for use case modelling in most UML CASE-tools is also limited. For example, they do not support traceability from use cases to other diagrams even though it is recommended in use case driven development.

Quality attributes of use case models and advice on how to construct them have been proposed in the literature [1-14]. These recommendations are mostly based on extensive experience from software projects. However, to our knowledge only the guidelines for use case authoring developed in the CREWS project [14] have been subject to empirical evaluation.

Use case models are frequently claimed to be easy to understand for the stakeholders involved in a development project [1-3], and a good understanding of the use case model is important if the use case model is to contribute successfully to the development project. In our opinion, understandability is therefore an important quality in use case models. Moreover, the large number of, sometimes contradictory, guidelines for use case modelling to choose from motivate the overall objective of this study: To empirically investigate the effect of different guidelines on the quality, in particular understandability, of a use case model.

We conducted an experiment with 139 students divided into 31 groups. The groups were organized in pairs; one group was the customer for a system to be developed, while the other group was the development team. The development teams used one out of three different sets of guidelines in the construction of a use case model for the system. The use case models were evaluated by the authors of this paper according to a

set of quality attributes. To evaluate understandability, the students answered questions about the functionality in the use case models. The students were also asked about how useful they found the guidelines.

The remainder of this paper is organized as follows. Section 2 gives an overview of different guidelines and recommendations on use case modelling together with suggestions for properties of quality. Section 3 describes our experiment in details. Our results are presented in Section 4 and discussed in Section 5. Section 6 discusses threats to the validity of the results. Ethical considerations relevant to this experiment are discussed in Section 7. Section 8 concludes and describes future work.

## 2 Use Case Modelling

Use cases can be described in many ways ranging from an informal, unstructured style to a more formal style approaching pseudocode [16]. Different organizations construct and apply use case models differently [13]. Independently of the format and notation, deciding a suitable level of detail is a challenge. A too fine granularity makes the use case model difficult to grasp, while a too coarse granularity hides the complexity [2].

Cockburn [5] recommends that the format should be chosen for each project, and that the choice should be driven by both characteristics of the development team and the main purpose of the use case model. The level of experience in the team, both regarding application domain and regarding use case modelling, is a relevant factor because little experience may require more support from the use case format. Another relevant factor is the cohesiveness of the team since a team working closely together can write more casual use cases than a larger, perhaps geographically dispersed team.

The future use of a use case model in a development project should also be an important issue when determining the appropriate format. An example is applying use case models in estimating effort for software development projects [6,17,18]. We have conducted a case study that indicates that the format of the use case model impacts the estimates [18].

Different formats may also support other activities differently, for example, the ability to identify classes during design or the ability to validate and verify an architectural decision.

There are, in our opinion, challenges at three levels when constructing a use case model. All three levels may be improved by appropriate guidelines. In our experiment, we test one guideline at each of the three levels, which are respectively described in Sections 2.1, 2.2 and 2.3. Section 2.4 describes some quality attributes of use case models.

### 2.1 Minor Guidelines

Actors are identified by considering who will receive information from a system and who will provide it with information. Use cases are identified by asking what are the main tasks of each actor. The first set of guidelines used in our experiment, called Minor guidelines (alternative 1 in Appendix A), are based on those found in [1,6].

They describe how to identify actors and use cases, but give little direction on how to construct them.

We included this alternative in our experiment primarily because we wanted to investigate how the students documented their use case models when they were not given specific guidelines on how to do it. Another purpose was to study how a more elaborate description of how to identify actors and use cases affected the resulting use case models.

## 2.2 Template Guidelines

A template is often recommended for documenting use cases because the predefined structure of a template forces the developer to identify and include important elements in each use case [2]. Examples of templates can be found in [2,5,6,10,11,12]; their most typical content is shown in Table 1. Our second set of guidelines is thus based on templates. The Template guidelines (alternative 2 in Appendix A) include a template for describing actors and a template for describing for use cases. These templates are based on the templates used in [2,5,6].

**Table 1. Content of templates**

| Property | [2] | [5] | [6] | [10] | [11] | [12] |
|---|---|---|---|---|---|---|
| Title | x | x | x | x | x | x |
| Actor(s) | | x | x | x | x | x |
| Trigger | x | x | | | | |
| Scope | | x | | | | |
| Summary | x | | x | x | x | x |
| Preconditions | x | x | x | x | x | x |
| Basic flow of events | x | x | x | x | x | x |
| Extension points | x | x | x | x | x | |
| Alternate courses | x | x | x | x | x | x |
| Post-condition | x | x | x | | x | x |

## 2.3 Style Guidelines

There are different recommendations on how to structure the description of the flow of events. In [1] and [8], narrative text with alternatives and extensions is recommended. Cockburn [5] recommends that users should warm up with narratives, but that the final use cases should follow a predefined template, as narrative use cases are often ambiguous and lack structure.

The guidelines described in [14] focus on the individual events in the flow of events. The aim is to give each description of events a standard structure and thereby make them easier to read. There seems to be an agreement on the following advice on how to describe the flow of events and each single event:

- Each event should be numbered in order to show how the process moves forward.
- Each event should clearly show which agent is active.
- Each event should be described with a present tense verb phrase in active voice.
- The user's vocabulary should be used.
- Repetition in the flow of events should be handled with a *while*, *for* or *repeat..until* construction.

However, there is some disagreement on the following:

- Whether the flow of events should be described in one section with narrative text, as a numbered list of events or as a list of event-response pairs.
- The handling of alternatives. In [6] and [14] an *if..then* construction is recommended, while such a construct is warned against in [2].
- The use of pseudocode in the description of each event. The structure recommended by Ben Achour *et al*. [14] strongly resembles pseudocode, while Kulak and Guiney [2] claim that pseudocode is too different from the user's language.

We have used a modified version of the guidelines described in [14], which we call Style guidelines[1] (alternative 3 in Appendix A) as the third set of guidelines in our experiment. We have modified the guidelines slightly based on the results from the experiment described in [15]. That experiment indicated that some of the original guidelines were difficult to use since they were implemented by few of the participants, and some were superfluous since all the participants, including those who did not receive the guidelines, implemented them. We decided to use these guidelines because they had been subject to evaluation in two former controlled experiments and could thus be evaluated in a different context in our experiment.

## 2.4 Recommendations for Use Case Models

It is believed that the quality of use case models has an impact on the quality of other documents subsequently produced in the development process. Hence, it seems sensible to investigate which properties of a use case model contribute to its quality. Many recommendations exist, for example, the following found in [2,8,12-14]:

- The use case should be easy to read.
- The descriptions should not include any assumptions of design or implementation.
- The descriptions should not include interface details.
- Events that are not related to the overall goal of the use case should not be described.
- The action descriptions should be complete.
- The flow structure should be correct and unambiguous.
- The terminology should be consistent.

[1] The original proposers of the guidelines called them guidelines for style and content. For simplicity reasons we will denote them just Style guidelines.

We have used these recommendations as a basis for the attributes against which the use case models in our experiment are evaluated.

# 3 The Experiment

This section describes the participants in our experiment, how they were trained, the detailed procedure of this experiment and the marking scheme. Finally, the hypotheses are presented.

## 3.1 Experiment Participants

The experiment was conducted as part of a compulsory project in an undergraduate course in software engineering with 139 students divided into 31 groups. We conducted a survey to identify the students' background.

The students were mostly aged between 20 and 30. Out of the 139 students, 43 had some experience from professional software development, while 9 had a lot of experience. There were 20 students who had professional experience with requirements engineering, 4 of them had done use case modelling. None of the students were familiar with any of the guidelines used in this experiment.

The students with experience in either project management or the languages used in the course (UML and Java) were *evenly* distributed among the groups. The other students were *randomly* assigned to the groups. Making the groups as similar as possible was important to study the effects of the guidelines.

The groups were organized in pairs with one group having the role of customer for the system to be developed, while the other group had the role of development team. Each group participated in two pairs, in one pair as customer for either system A or B (see appendix B) and in the other pair as development team for the other system. There was one exception; one group was both customer and developer for system A because of the odd number of groups.

## 3.2 Training

The software engineering course consisted of lectures and seminars. The students were divided into six seminar groups led by graduate students. Requirements engineering was taught in a one-hour lecture; basic use case modelling was taught in another one-hour lecture and in a one-hour seminar. The students were taught the concepts of actors and use cases, and they were given two examples (the same two for all groups) on how to describe use cases and their flow of events. In another one-hour seminar, the students were taught the guidelines and examples of how to use them.

### 3.3 Procedure of Experiment

The first activity in the student project consisted in creating an informal requirements specification based on the description of the system for which they were customers. This requirements specification was then handed on to their development team. The development teams made a detailed requirements specification including a use case model. The groups had two weeks available for this task, and they had the opportunity to ask lecturers, seminar leaders and fellow students for help. It was recommended that they should talk to their customer in order to clarify their requirements.

When the requirements specification was completed, it was made available to the customer group. Some groups received use case models written using the same guidelines as they had used themselves, while others received use case models written using different guidelines.

The students then individually answered a questionnaire (shown in appendix C) with questions about the functionality in the use case model they had made themselves and about the functionality in the use case model they received from their development team. The questions were constructed based on the original system description. The questionnaire also included a question about how useful they found the guidelines, which was answered by ticking one of the following alternatives: *Very useful*, *quite useful, quite useless* or *very useless*. There were also questions about how much time they had spent working on both the informal and the formal specification, on communication with their development team and on reading through the use case model from the development team. The students had previously been asked to record effort so it should be possible for them to recapture the time used. It was compulsory to answer the questionnaire, and it was done at times normally scheduled for seminars. The students were given directions on how to answer the questions, and one of the authors was available for questions when they answered the questionnaires.

Answers to questions about functionality were used in another experiment to evaluate differences in understandability between requirements expressed in natural language and requirements expressed in an activity diagram [19]. In that experiment the participants were asked to tick the correct answer to a question from a list of possible an answers. In our experiment it was infeasible to give the participant a predefined list of alternative answers because we did not know the actual use cases beforehand. Instead the participants answered using free text. The questionnaire included two blank lines after each question to give the participants an indication of the length expected for their answers.

When answering the questionnaire, all the students had available a copy of their own requirements specification and the one made by their development team. On average they spent 40 minutes on these questions.

### 3.4 Marking Scheme

The resulting use case models were evaluated according to a list of different properties inspired both by the recommendations on how to write use cases described in Section 2 and by the marking schemes for flow of events suggested in [14,15].

Ben Achour *et al.* [14] used the following marking scheme in their experiment for evaluating the CREWS guidelines:

- Completeness for each event, determined by counting the amount of events that included agents, objects, communication sources and destination
- Completeness of the whole flow of events, determined by counting irrelevant action descriptions compared with a use case made by an expert on the application domain (a low count gave a high score)
- Number of correctly placed variations
- Number of homonyms and synonyms (a low count gave a high score)

In our experiment there were very few missing elements in the descriptions, so we decided not to include this aspect. Since our use case models were constructed from different informal specifications, we did not have the opportunity to compare the individual use cases with ideal solutions, but in our opinion there were very few irrelevant descriptions of events. We did, however, find missing or unrealistic descriptions, so the realism of the description of the flow of events was also considered in our evaluation. There were significant differences in how the students handled variations in the use cases. We have therefore counted both the number of variations for each use case model and the number of correct placements in the flow of events. The students used inconsistent terminology in the use case models. We therefore also included this aspect in our evaluation.

Cox & Phalp [15] describe a replication of the experiment conducted by Ben Achour *et al*. [14]. They found the original marking scheme difficult to use because it was too detailed to give a good overall assessment of the individual use cases. In addition to the marking scheme above, they therefore also evaluated the use cases more subjectively according to:

- Plausibility – the realism of the use case
- Readability – the flow of the use case
- Consistent structure – consistent terminology and use of present simple tense
- Alternative flow – consideration of variations

We found all these aspects relevant and therefore included them in our evaluation. In the two experiments described above, only single use cases were evaluated. In our experiment, complete use case models were constructed, each containing several use cases. Since we considered also aspects of the overall model, and because of the large number of use cases in our experiment, we used a slightly different marking scheme, which is based on a more overall evaluation of the use cases:

- Single diagram – the use case model should include one single diagram showing all the actors and use cases.
- Actors – the correct actors were identified. Correctness was determined relative to the informal requirements specification.
- Use cases – the correct use cases were identified. Correctness was determined relative to the informal requirements specification as above.
- Content – the description of each use case contained the information required by all the sets of guidelines: actor, assumptions that must be valid before the use case starts, flow of events, variations and post-conditions.

- Level of detail – the descriptions of each event were at an appropriate level of detail. There should be no unnecessary details about user interface or internal design. Each event should be atomic, that is, sentences with more than two clauses should be avoided.
- Realism – the flow of events was realistic, that is, the events follow a logical and complete sequence, and it is clearly stated where variations can occur.
- Consistency – the use of terminology was consistent.

**Table 2. Properties supported by guidelines**

| Type of guideline | Single diagram | Actors | Use cases | Con-tent | Level of detail | Rea-lism | Con-sistency |
|---|---|---|---|---|---|---|---|
| Minor guidelines | x | x | x | | | | |
| Template guidelines | x | | | x | | | |
| Style guidelines | x | | | | x | x | x |

**Table 3. Marking scheme**

| Property | Mark | Comment |
|---|---|---|
| Single diagram | 0-1 | 1 = correct, 0 = wrong |
| Actors | 0-3 | 3 = all correct, 0 = all wrong |
| Use cases | 0-3 | 3 = all correct, 0 = all wrong |
| Content | 0-3 | 3 = all correct, 0 = all wrong |
| Level of detail | 0-3 | 3 = all correct, 0 = all wrong |
| Realism | 0-3 | 3 = all correct, 0 = all wrong |
| Consistency | 0-2 | 2 = all correct, 0 = all wrong |

The guidelines gave different support for these properties. Table 2 shows which guidelines support which properties. Each use case model was given a mark for each of these properties based on an overall assessment according to the marking scheme in Table 3.

The size of the use case models is also measured. Size is measured as a vector:

< *Number of actors,*
*number of use cases,*
*median number of actions in the flow of events,*
*median number of variations* >

We believe that the number of identified actors and use cases, together with the number of events and variations, indicate quality of the guidelines – the higher number, the better quality.

## 3.5 Hypotheses

This section presents the hypotheses tested in this experiment. Our first hypothesis ($H1_1$) is that different guidelines for constructing use case models have different effect on how well the use case models are understood by their readers.

**$H1_0$**: There *is no* difference in understanding when reading use case models constructed with different guidelines.
**$H1_1$**: There *is* a difference in understanding when reading use case models constructed with different guidelines.

Our second hypothesis ($H2_1$) is that the different guidelines have different effect on the understanding of the requirements from the point of view of those who use the guidelines to construct use case models.

**$H2_0$**: There *is no* difference in the understanding of the requirements when using different guidelines in the construction of use case models.
**$H2_1$**: There *is* a difference in the understanding of the requirements when using different guidelines in the construction of use case models.

Our third hypothesis ($H3_1$) is that the different guidelines are of different usefulness to those who construct the use case models.

**$H3_0$**: There *is no* difference in the usefulness of the different guidelines when constructing use case models.
**$H3_1$**: There *is* a difference in the usefulness of the different guidelines when constructing use case models.

# 4 Results

This section describes the results from the testing of the hypotheses. We used the non-parametric Kruskal-Wallis test since our data sets were not normally distributed. Results of the effect of the guidelines on other quality attributes are also presented.

## 4.1 Assessment of Understandability

After reading the completed questionnaires, we found that many of the questions about the functionality were irrelevant for the use case models describing system A. The reason was that many of them included other functionality than we had expected from the original description (Appendix B). We therefore consider only system B in the analysis of the answers about functionality.

The answers to the questions about functionality were compared with the functionality actually described in the use case models in order to determine their correctness. Each answer was given a mark of 0 (wrong answer or no answer), 1 (correct answer to a simple question or partially correct answer to a complicated question) or 2 (correct answer to a difficult question). Questions 1, 2 and 7 for system B (Appendix C) were classified as simple. The answers for system B could obtain a maximum of 13 points.

An example of a partially correct answer is when question B.2, "How is the roster made and updated, and who is responsible for it," has the answer "unit nursing officer." This answers the second part of the question but not the first part. The

guidelines were compared regarding the score on correct answers for each individual customer and individual developer.

Table 4 shows the number of students who read the use case models for system B, distributed by the guidelines used in the construction of these models; similarly for the use case model construction. This table also shows the minimum, median, maximum and standard deviation of the scores on correct answers given by the readers and constructors, respectively.

**Table 4 Descriptive statistics on the data used in the assessment of understandability**

| Type of guideline | Reading | Scores on reading | | | | Constructing | Scores on constructing | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Min | Median | Max | Std | | Min | Median | Max | Std |
| Minor guidelines | 14 | 2 | 6 | 11 | 2,6 | 13 | 5 | 8 | 12 | 2,4 |
| Template guidelines | 26 | 5 | 9 | 12 | 2,1 | 25 | 4 | 9 | 12 | 2,5 |
| Style guidelines | 27 | 1 | 8,5 | 13 | 2,9 | 28 | 2 | 9 | 13 | 2,7 |
| Total | 68 | | | | | 66 | | | | |

**Reading Use Case Models – Hypothesis H1.** There was a significant difference in the score on correct answers between the customers who had read use case models constructed by developers who had used either the Template or Style guidelines compared with those who had used the Minor guidelines (Figure 1). There was no significant difference between the Template and Style guidelines, although the Template guidelines were slightly better.

The level of significance (alpha-level) chosen for this test was 0.05. The p-value of 0,021 obtained from the test signifies that we can reject $H1_0$ and that we can assume with a 95% probability that there is a difference between the guidelines.

```
Guidelines     N     Median     Ave Rank          Z
Minor         14      6,000         22,2     -2,61
Template      26      9,000         40,1      1,85
Style         28      8,500         35,4      0,32
Overall    68                     34,5

H = 7,58  DF = 2  P = 0,023
H = 7,70  DF = 2  P = 0,021 (adjusted for ties)
```

**Fig. 1. Kruskal-Wallis test on correct answers for customers who had read use case models for system B**

Half of the customers read use case models constructed using the same guidelines as they had used themselves in the role of developers. The other half read use case models constructed with guidelines with which they were unfamiliar. We investigated whether there was a difference in understanding related to whether the guidelines were familiar.

When the guidelines were familiar, there was a significant difference in favour of the Template guidelines (Figure 2). However, the number of subjects for the Style guidelines was much higher than for Minor or Template guidelines (24 vs. 5 and 6). This may explain why the Style guidelines did worse than when all the customers for System B were considered.

| **Guidelines** | **N** | **Median** | **Ave Rank** | **Z** |
|---|---|---|---|---|
| Minor | 5 | 9,000 | 15,3 | -0,64 |
| Template | 6 | 11,000 | 28,8 | 2,84 |
| Style | 24 | 8,000 | 15,9 | -1,83 |
| Overall | 35 | | 18,0 | |

H = 8,11 DF = 2 P = 0,017
H = 8,49 DF = 2 **P = 0,014** (adjusted for ties)

**Fig. 2. Kruskal-Wallis test on correct answers for customers who had read use case models for system B constructed with the guidelines they had used themselves**

When the guidelines were unfamiliar, the Style guidelines apparently did well (Figure 3). However, this sample is very small (4), which means that we should not draw any conclusions on the effect of the Style guidelines from this test. We repeated the test with only the Minor and Template guidelines, and found a significant difference (p = 0,004) in favour of the Template guidelines.

| **Guidelines** | **N** | **Median** | **Ave Rank** | **Z** |
|---|---|---|---|---|
| Minor | 9 | 6,000 | 8,2 | -3,19 |
| Template | 20 | 8,000 | 18,7 | 1,25 |
| Style | 4 | 11,000 | 28,3 | 2,48 |
| Overall | 33 | | 17,0 | |

H = 13,45 DF = 2 P = 0,001
H = 13,73 DF = 2 **P = 0,001** (adjusted for ties)

* NOTE * One or more small samples

**Fig. 3. Kruskal-Wallis test on correct answers for customers who had read use case models for system B constructed with different guidelines than they had used themselves**

**Understanding Requirements – Hypothesis H2.** There were no significant differences between the guidelines when we compared the scores of the developers on the questions about functionality in the use case models they had constructed themselves (Figure 4). This indicates that the understanding of the requirements among those who had developed a use case model, depends primarily on other factors than the guidelines used when they constructed the use case model. The p-value of 0,835 obtained from the test indicates that we cannot reject $H2_0$.

| Guidelines | N | Median | Ave Rank | Z |
|---|---|---|---|---|
| Minor | 13 | 8,000 | 31,0 | -0,53 |
| Template | 25 | 9,000 | 33,4 | -0.04 |
| Style | 28 | 9,000 | 34,8 | 0,47 |
| Overall | 66 | | 18,0 | |

H = 0,35 DF = 2 P = 0,838
H = 0,36 DF = 2 **P = 0,835** (adjusted for ties)

**Fig. 4. Kruskal-Wallis test on correct answers for developers who had constructed use case models for System B**

## 4.2 Assessment of Usefulness – Hypothesis H3

The questionnaire given to the students also included a question about how useful they found the guidelines. This question was answered by ticking one of four alternatives and each alternative was given a mark: *very useful* = 3, *quite useful* =2, *quite useless* =1 and *very useless* = 0. This question was equally relevant to both system A and B. All the questionnaires were therefore included in this analysis. This gave a total of 138 subjects (one of the students did not answered this question).

The p-value of 0,113 obtained from this test gives a weak indication that the Template guidelines were found most useful. We repeated the test with only the Minor and Template guidelines, and found a significant difference in favour of the Template guidelines (p=0,039). This indicates that we can reject $H3_0$.

| Guidelines | N | Median | Ave Rank | Z |
|---|---|---|---|---|
| Minor | 33 | 2,000 | 61,7 | -1,29 |
| Template | 44 | 2,000 | 77,8 | 1,68 |
| Style | 61 | 2,000 | 67,7 | -0,46 |
| Overall | 138 | | 69,5 | |

H = 3,31 DF = 2 P = 0,191
H = 4,37 DF = 2 **P = 0,113** (adjusted for ties)

**Fig. 5. Kruskal-Wallis test for developers on how useful they found the guidelines they had used**

## 4.3 Assessment of Quality

The use case models constructed by the students in our experiment were evaluated according to the marking scheme described in Table 3, Section 3.4. Table 5 shows that use case models constructed using the Template guidelines obtained the highest overall score on the quality attributes. The developers using the Minor guidelines were best at understanding that they should make one single diagram, and they did better than those using the Style guidelines on identifying the correct actors and use cases. The use case models constructed using the Style guidelines obtained the highest score on consistency, that is, consistent use of terminology, and they did quite well on level of detail and realism.

**Table 5. Average score on the properties of quality**

| Type of guideline | Single diagram | Actors | Use cases | Content | Level of det. | Realism | Consistency | Sum |
|---|---|---|---|---|---|---|---|---|
| Minor guidelines | Best (0,9) | Mid (2,3) | Mid (2,1) | Worst (1,1) | Worst (1,7) | Worst (1,7) | Mid (1,7) | Worst (11,3) |
| Template guidelines | Mid (0,8) | Best (2,6) | Best 2,5 | Best (2,5) | Best (2,2) | Best (2,4) | Mid (1,7) | Best (14,6) |
| Style guidelines | Worst (0,6) | Worst (2,2) | Worst (1,8) | Mid (1,7) | Mid (1,9) | Mid (2,0) | Best (1,8) | Mid (12,0) |

Table 6 shows the size of the use case models constructed using the different guidelines. The fields of the size vector contain the median value (see Section 3.4). The use case models constructed using the Minor guidelines included on average the largest number of actors and use cases, but the lowest number of events in each use case. Only one of the use case models constructed with these guidelines included variations. The use of the Template and Style guidelines resulted in use case models of approximately equal size. However, the use case models constructed with the Style guidelines had slightly more use cases, while those constructed with the Template guidelines included more variations.

Table 7 shows some typical mistakes committed by the subjects in this experiment related to each of the quality attributes.

**Table 6. Median size for the use case models**

| Type of guideline | Size |
|---|---|
| Minor guidelines | <3,8,2,0> |
| Template guidelines | <2,5,5,2> |
| Style guidelines | <2,6,5,1> |

**Table 7. Typical mistakes in the use case models**

| Property | Typical mistake(s) |
|---|---|
| Single diagram | – Splitting the use case model into several diagrams, one diagram for each single use case. |
| Actors | – Omitting external systems.<br>– Including several actors who have exactly the same goals when using the system and who should therefore have been a single actor. |
| Use cases | – Including auxiliary functions that were not part of the requirements, overlooking use cases.<br>– Splitting events relating to the same goal on several use cases. |
| Content | – Omitting assumptions or result. |
| Level of detail | – Describing what happens inside the system or the user interface.<br>– Giving a brief and too incomplete description. |
| Realism | – Including common functionality in several use cases when it should have been separated out as a use case on its own.<br>– Omitting variation or neglecting to state where in the flow of events they can occur. |
| Consistency | – Using different words for the same entity. |

# 5 Discussion

This section discusses how the different guidelines affected the understandability and quality in our experiment.

## 5.1 Minor Guidelines

The Minor guidelines contained the most elaborate description of how to identify actors and use cases. The use case models constructed using these guidelines included, on average, the largest number of actors and use cases. However, they did not receive the highest score on correct actors and use cases, which indicates that some of their actors and use cases were superfluous. These guidelines also received the lowest overall score on the quality attributes. In our opinion, this indicates that these guidelines gave insufficient support on how to document actors and use cases, because good support for documenting use cases would have helped remove the superfluous actors and use cases.

The groups who used the Minor guidelines seemed to have more problems following the guidelines than the other groups. The use case models constructed using these guidelines did significantly worse on understandability than the use case models constructed with the other guidelines. We believe the reason was that these use cases lacked a coherent structure. The students participating in this experiment also found these guidelines the least useful.

## 5.2 Template Guidelines

The results in Section 4.1 indicate that use case models constructed using the Template or Style guidelines are easier to understand than use case models constructed using the Minor guidelines. The tests on the scores on correct answers from the customers also give a weak indication that the Template guidelines are better than the Style guidelines. We believe that the structure imposed by the Template guidelines makes it easier to find information in these use case models.

The Template guidelines also did better than the other guidelines regarding different quality attributes of the use case models. In our opinion, this may indicate a relationship between those attributes and the understandability. For example, the groups that followed the Template guidelines handled variations better than the other groups. This is an important aspect since the basic flow of events is often well known, but the alternatives to the basic flow are often not thought of.

These groups followed the guidelines most closely, which indicates that the guidelines are easy to use. The Template guidelines were also considered significantly more useful than the Minor guidelines and slightly more useful than the Style guidelines. We believe that these differences may be due to the templates being easier to understand and apply because it is made explicit what information should be inserted. In our opinion, the support from a template may be particularly important in an environment where the developers have little experience with the application domain and therefore need to work a lot on the requirements to develop a good

understanding. We also believe that the template is particularly useful for developers who have little experience with use case modelling.

### 5.3 Style Guidelines

Two experiments have previously been conducted to evaluate the Style guidelines regarding how they contribute to completeness, correctness and consistency in the use cases [14,15]. The results from the first experiment showed that the guidelines were usable and helpful in improving the use cases regarding those properties. (These guidelines were proposed by the same research group that conducted the experiment.) However, these results were not confirmed when the experiment was replicated [15], but both agree that the guidelines should be considered when authoring use cases.

In our experiment, the Style guidelines did almost as well as the Template guidelines when the understanding of the readers was compared, and they did better than the Minor guidelines on both quality attributes and usefulness. Hence, it appears that the Style guidelines did better in our experiment than in the experiment reported in [15]. This may be due to a different marking scheme or because the participants in our experiment had more time to thoroughly understand and apply the guidelines. We do not believe that the modifications we made to the original Style guidelines invalidate a comparison, because the changes we made consisted in removing parts of the guidelines that were not applied by the subjects in that experiment. In our opinion, this indicates that Style guidelines may successfully supplement Template guidelines.

## 6 Threats to Validity

This study is exploratory in the sense that we do not know any other studies where different guidelines on use case modelling have been compared. We would call it a semi-controlled experiment because it was done as part of a course, and thus we had not full control over all parts of the study. For example, we could not control the informal specifications from which the groups constructed the use case models. Moreover, we do not know in details how the students worked when constructing the use case models.

Determining how to analyze the results was a challenge, in particular regarding correctness of the answers to the questionnaires and the different attributes of the use case models. On the other hand, we believe that the organization in customer and developer groups contributed to a realistic setting. The next sections describe factors that we believe may have influenced our results.

### 6.1 Students as Subjects

Our experiment was conducted with students as subjects. It is therefore uncertain to what extent our results can be generalized to an industrial setting. However, many of the students in this experiment were experienced software developers who work part-time or previously worked in industry. Out of 139 students, 43 answered that they had

*some* relevant experience, 9 answered that they had *a lot of* experience. In our opinion, a large percentage of the students may therefore be comparable to professional software developers with up to two years experience.

Use case modelling is a relatively new technique; many organizations are currently starting to use it. Hence, it is not uncommon that developers have none or only a little experience with use case modelling. So, typical professionals may not have much more experience with use case modelling *per se* than our students, but on the other hand, our students are less experienced with requirements and with modelling in general than are professionals.

Høst *et al.* [20] compared the results from students and professional software developers in a study on factors affecting the lead-time of software development projects. They did not find significant differences between the answers from the students and the professionals, even though knowledge of factors affecting lead-time appears to depend on experience from software development projects.

To overcome the difficulties of using students as subjects, we plan to carry out a similar experiment with 20 professionals as subjects in a four-hour experiment. The explorative nature of this study was an important reason why we chose to conduct an experiment with students as subjects even though they may not be fully compatible with professional developers. Therefore, we found it necessary to test our experimental design on students and perhaps eliminate some hypotheses before conducting an experiment with professionals, as recommended by Tichy [21].

### 6.2 Complexity of the Task

The use case models constructed in our experiment were smaller than most use case models describing real systems. Therefore, we do not know if our results are applicable when the use case models are considerably larger. We conducted a case study in industry on how to apply use case models in estimation [18]. One of the two software development projects used in that case study was characterized as medium sized. That project lasted 7 months, and the use case model consisted of 7 actors and 9 use cases, which is not substantially more than the number of actors and use cases in the use case models in the experiment reported in this paper. However, the number of events and variations was considerably higher than in our experiment. We therefore believe the use case models in our experiment may be comparable to use case models in small industrial projects.

Our guidelines did not handle secondary actors nor included and extending use cases, and alternative flows of events were handled superficially. This may also mean that the use case models in our experiment are not comparable to use case models for real systems on all aspects. We used quite simple guidelines because we wanted to limit the number of attributes of the use case models that we would have to consider. Again, the reason for this decision was the exploratory nature of our research. Nevertheless, we plan to conduct further studies using extended guidelines.

The differences in size among the use case models in our experiment may have influenced the correctness of the answers about functionality. A small use case model consisting of a small number of use cases described with few details may have made it easy to answer the questionnaire correctly despite low quality of the use model.

Correspondingly, very detailed use case models may be time consuming to read, which in turn may lead to wrong answers even though they are of high quality. Another aspect of this, is that a high quality use case model may have appeared very convincing to the customer group, and thereby led them to believe that all their requirements were included even if they were not.

### 6.3 Participants both as Customers and Developers

The organization into customer and developers meant that the readers of the use case models all had experience with writing use cases. This will not always be the case for all the stakeholders in a real project. Therefore, we cannot be sure that the result indicating that the use case models constructed using Template guidelines are the easiest to understand, is applicable to customers who are unfamiliar with use case modelling.

However, it is recommended that use case models should be constructed together with their future users. To enable the future users to participate in the use case modelling process, it is common to train them in basic use case modelling. Many customers therefore have some knowledge of use case modelling.

### 6.4 Motivation

In a large course like our software engineering course, there will inevitably be differences in motivation among the students. We observed differences in motivation regarding answering the questionnaires since the seminars where this was done were compulsory. The students were not used to attendance being compulsory, so some were quite negative about that.

### 6.5 Dependence on Informal Specifications

The use case models were constructed from informal specifications made by the customer groups. The informal specifications had varying quality, and there were differences in how closely they were followed by the development team. In the cases where the informal specification covers all the information in the use case model, the students may have been able to answer the questionnaire correctly without having understood the use case models. Moreover, although the students were given an explanation of how to answer the questionnaire, some may not have understood exactly on what they should base their answers.

### 6.6 Experience

Experience with use case modelling may lead to higher quality of the use case models independently of the guidelines. Experience with the application domain might in our experiment have affected the answers to the questionnaires as it may lead to expectations regarding the functionality [22]. However, there were few subjects with

application domain experience. We believe that differences in experience did not affect the results since the students were randomly assigned to the groups.

### 6.7 Questionnaires

Some of the groups made specifications with functionality that was very different from what we expected to be the outcome when we wrote the system descriptions. This lead to some of the questions being irrelevant for some of the use case models. The questions generally seem to have been better suited for system B than for system A. Questions made specifically for each use case model would probably have made it easier to determine how well the use case model was understood. This was infeasible due to the large number of groups and that we wanted the students to fill in the questionnaires shortly after the use case models were completed.

The correctness of the answers was determined subjectively. This may represent a source of error, as it was not always obvious what the correct answer should be. However, the use case models were simple and the answers given were a maximum of two lines of text, so in most cases determining whether an answer was correct was relatively easy.

## 7 Ethical Considerations

Due to the project being compulsory, the workload on the groups should be approximately even. This meant that all the groups had to use some kind of guidelines, and that learning and implementing them should be equally work consuming. It should also be ensured that all the students had the opportunity to learn equally much.

In our experiment, we attempted to achieve not too large differences between the groups of students by not making the guidelines too different. Of course, this concern made it more difficult to observe the different effects of the various guidelines.

The results from the experiment were presented to the students in a one-hour lecture to give all the students a flavour of all the guidelines. As an afterthought, we realize that this was probably insufficient to ensure that all the students learned equally much. If such an experiment is to be repeated, we would recommend that all the students are given exercises including all the guidelines.

The students were in the questionnaire encouraged to give feedback on how they felt about the experiment. Most of the students were positive, for example, they reported that they through this experiment learned more about use case modelling than they would have done otherwise.

## 8 Conclusions and Future Work

We have identified three categories of guidelines for use case modelling, and we have conducted an experiment with the aim of detecting the effects of using them. The results from the experiment indicate that guidelines based on templates support the construction of use case models that are easier to understand for the readers than guidelines without specific details on how to document use cases. The guidelines based on templates were also considered by the participants as the most useful when constructing use cases. Our experiment further indicates that the guidelines based on templates result in better use case models regarding also other quality attributes.

Style guidelines focus on the documentation of the flow of events of each use case. They appear to have contributed to some of the quality attributes. Therefore, it may be beneficial to combine the template guidelines with style guidelines.

This study was exploratory. We will use the results as a basis for further studies both on how to improve the ease of understanding use case models and on how they should be used in subsequent phases of a development project. The following research activities are planned:

- A replication of this experiment using modified versions of the guidelines presented in this paper. The modifications will be based on the results from this experiment and on the extensions suggested in Section 6.2. We also intend to investigate the effects of the different guidelines on the understanding of the groups as a whole, by letting the groups answer the questionnaires instead of the individual participants.
- A follow-up controlled experiment similar to the one reported in this paper, but this time in industry with professional software developers.
- A case study on the application of use case models in software development projects in industry, in particular on how use case models can be used in estimating software development effort.
- A field experiment on how different stakeholders in a project understand use case models and how a reading technique may improve it [22].

## Acknowledgements

We thank Lars Bratthall for useful comments on earlier versions of this paper and Erik Arisholm for guidance on the statistics. We also thank all the students who participated in the experiment. This research is funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry).

## References


1. Jacobson, I. *et al.* Object-Oriented Software Engineering. A Use Case Driven Approach. Addison-Wesley, 1992.
2. Kulak, D. & Guiney, E. Use Cases: Requirements in Context. Addison-Wesley, 2000.
3. Booch, G., Rumbaugh, J. & Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley, 1999.
4. Cockburn, A. Structuring Use Cases with Goals. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, Ha.T.TR.95.1, http://members.aol.com/ acockburn/papers/usecases.htm, 1995.
5. Cockburn, A. Writing Effective Use Cases. Addison-Wesley, 2000.
6. Schneider, G. & Winters, J. Applying Use Cases – A Practical Guide. Addison-Wesley, 1998.
7. Constantine, L. L. & Lockwood, L. A. D. Software for Use. A Practical Guide to the Models and Methods for Usage-Centered Design. Addison-Wesley, 1999.
8. Rosenberg, D. & Scott, K. Use Case Driven Object Modelling with UML. Addison-Wesley, 1999.
9. Regnell, B., Andersson, M. & Bergstrand, J. A Hierarchical Use Case Model with Graphical Representation. Proceedings of Second IEEE International Symposium on Requirements Engineering (RE'95), York, UK, 1995.
10. Harwood, R. J. Use case formats: Requirements, analysis, and design. Journal of Object-Oriented Programming, Vol. 9, No. 8, pp. 54-57, January 1997.
11. Mattingly, L. & Rao, H. Writing Effective Use Cases and Introducing Collaboration Cases. Journal of Object-Oriented Programming, Vol. 11, No. 6, pp. 77-79, 81-84, 87, October 1998.
12. Jaaksi, A. Our Cases with Use Cases. Journal of Object-Oriented Programming, Vol. 10, No. 9, pp. 58-64, February 1998.
13. Firesmith, D.G. Use Case Modeling Guidelines. Proceedings of Technology of Object-Oriented Languages and Systems – TOOLS 30. IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.
14. Ben Achour, C., Rolland, C., Maiden, N.A.M. & Souveyet, C. Guiding Use Case Authoring: Results of an Empirical Study. Proceedings IEEE Symposium on Requirements Engineering, IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.
15. Cox, K. & Phalp, K. Replicating the CREWS Use Case Authoring Guidelines. Empirical Software Engineering Journal, Vol. 5, No. 3, pp. 245-268, 2000.
16. Hurlbut, R.R. A Survey of Approaches for Describing and Formalizing Use Cases. Technical Report: XPT-TR-97-03, Expertech, Ltd., 1997.
17. Martinsen, S.A. & Groven, A-K. Improving Estimation and Requirements Management Experiences from a very small Norwegian Enterprise. SPI 98 Improvement in Practice: Reviewing Experience, Previewing Future Trends. The European Conference on Software Improvement. Meeting Management, Farnham, UK, 1998.
18. Anda, B., Dreiem, H., Sjøberg, D. & Jørgensen, M. Estimating Software Development Effort Based on Use Cases – Experiences from industry. Submitted to UML'2001 (Fourth International Conference on th Unified Modeling Language).
19. Cioch, F.A. Measuring Software Misinterpretation. Journal of Systems and Software, Vol. 14, No. 2, pp. 85-95, February 1991.
20. Høst, M., Regnell, B. & Wohlin, C. Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empirical Software Engineering, Vol. 5, No. 3, pp. 210-214, November 2000.
21. Tichy, W.F. Hints for Reviewing Empirical Work in Software Engineering. Empirical Software Engineering, Vol. 5, No. 4, pp. 309-312, December 2000.
22. Anda, B. & Jørgensen, M. Understanding Use Case Models. Proceedings of Beg, Borrow, or Steal Workshop, International Conference on Software Engineering, June 5, 2000, Limerick, Ireland.

# Appendix A

This appendix contains the three sets of guidelines used in this experiment.

## Alternative 1 – Minor guidelines

The Use Case Model should include:

1. A use case diagram that shows actors and use cases
2. A description of each actor
3. A description of each use case

Below is a description of the process that you should follow when constructing the use case model.

**1. Start by identifying the actors**
An actor is an entity that interacts with the system. Actors can be:

- A human user
- Another system which receives services from this one
- Another system which offers services to this one
- Actors are external to the system

The first step in identifying actors consists in finding users, but remember that users =/= actors:

⇒ Identify the most important users of the system
  - For whom will the system be constructed?
  - Who receives information from the system?
  - Who supplies the system with information?
  - Who removes information from the system?

⇒ Identify other users
  - Which interactions will de done with other systems?
  - What external hardware is necessary?
  - Who performs administration and maintenance?

The second step consists in finding roles:

⇒ Find out what roles the users have (roles encapsulate the way the system is used, but remember that roles are not equivalent to users nor to job-descriptions)

⇒ An actor constitutes a single role

**2. Identify use cases and draw a use case diagram**

⇒ The use cases describes what the actors wishes to do with the system, that is the actors goals

⇒ Each goal is represented by a use case

⇒ Identify use cases by looking at

- What are the main tasks of each actor?
- Will the actor read/write or change some of the information in the system?
- Should the actor inform the system about changes happening outside the system?
- Does the actor wish to be informed about unexpected changes?

**3. Describe each actor**

⇒ Give a brief description of each actor with name and most important goals when using the system.

**4. Describe each use case in detail**

⇒ Describe the flow of events in the use case, that is, all the steps in the interaction between actor and system that are necessary for the actor to reach his goal.

⇒ The description should show:

- The actors input
- Objects (including actors) that are involved
- Assumptions that are made about the objects
- The result of the use case

## Alternative 2 – Template guidelines

The Use Case Model should include:

1. A use case diagram that shows actors and use cases.
2. A description of each actor according to the first template below.
3. A description of each use case according to the second template below.

**Template for describing an actor:**

| | |
|---|---|
| **Actor** <*name*> | |
| **Description** <*A short text that describes the actor*> | |
| **Examples** | |

**Template for describing a use case:**

| | |
|---|---|
| **Use Case** <*name*> | |
| **Actors** <*name*> | |
| **Trigger**<br><*The event which starts the use case*> | |
| **Prerequisites**<br><*Constraints that must be met for the use case to be executed*> | |
| **Post-conditions**<br><*Conditions which are met when the use case terminates*> | |
| **Normal flow of events**<br><*A simple, brief description of the series of events of the most likely outcome*> | |
| **Variations**<br><*Variations on the normal flow of events, why it is followed, and outcome*> | |
| **Use Case associations**<br><*A list of other use cases that this use case is extended by or is used by*> | |

## Alternative 3 – Style guidelines

The Use Case Model should include:

1. A use case diagram that shows actors and use cases.
2. A description of each actor with name and most important goals when using the system.
3. A description of each use case which shows
    - The actors input
    - Objects (including actors) that are involved
    - Assumptions that are made about the objects
    - The result of the use case

In addition to this each use case should show the flow of events in the use case. The flow of events consists of a number of actions, and each action should be described so that it satisfies the guidelines below.

Style guidelines-

**SG1**: Write the UC normal course as a list of discrete actions in the form: <action#><action description>. Each action description should start on a new line. Since each action is atomic, avoid sentences with more than two clauses.

**SG2**: Use the sequential ordering of action descriptions to indicate strict sequence between actions. Variations should be written in a separate section.

**SG3**: Iterations and concurrent actions can be expressed in the same section of the UC, whereas alternative actions should be written in a different section.

**SG4**: Be consistent in your use of terminology, that is, use consistent names on actors, objects and actions in all action descriptions. Avoid use of synonyms and homonyms.

**SG5**: Use present tense and active voice when describing actions.

**SG6:** Avoid use of negations, adverbs and modal verbs in the description of an action.

Guidelines for content-

**CG1**: <agent ><action><agent>

**CG2**: <agent><action><object> <prepositional phrase>

**CG3**: 'If' <alternative assumption> 'then' <list of action descriptions>

**CG4**: 'Repeat until' <repetition condition> <list of action descriptions>

**CG5**: <action 1> 'while' <action 2>

# Appendix B

This appendix includes the descriptions of the two systems for which the students should specify a use case model.

### Description – System A

An opinion poll institute want a system with questionnaires on the Internet. The system should make it easy to publish questionnaires, as well as to fill in questionnaires on the Web. The answers to the questionnaires should be saved so that they can be exported to other tools (an example is "structured text" which can be imported into a spreadsheet). For some of the questions it will be necessary for the system to read a significant amount of text. The opinion poll institute want an easy overview of the answers received, for example, they want to know how many have answered the different questions. Notice that you shall not make a simple questionnaire on the Web, but a "questionnaire generator" for the Web.

### Description – System B

A hospital ward needs a system for swapping duties between nurses. There will be a PC in the ward where the nurses can log in. The user interface should make it possible to register who swaps duties and for what period of time. First the swap is registered with status *inquiry*. When the head of the ward has accepted the swap, the status should be changed to *accepted*. Swaps that are not accepted should be registered with status *not accepted.* If the head of the ward has not responded to the inquiry within 24 hours, status should automatically be set to *accepted.* The nurses must be able to log on to the system to see if the swap of duties is accepted. All accepted duties should be saved so that they can be transferred to other systems.

# Appendix C

This appendix shows the questions about functionality in the use case models that were included in the questionnaires given to the students.

**Questions for System A:**

1. How many questions can there be in a questionnaire? If there is a limit to the number, on what is this limit based?
2. Which different alternatives are allowed for the answers?
3. Is it possible to insert comments either to questions or to answers in questionnaires?
4. Is there any validation of questionnaires that have been completed? If the answer is "yes", how is the validation done?
5. Who has access to the answers from a survey?
6. What possibilities are there in the system for analyzing answers and who has access to these possibilities?
7. To which tool can answers from questionnaires be exported and how is this done?
8. What possibilities exist for changing questionnaires that have already been saved, and who has access to these possibilities?
9. What possibilities exist for changing answers or continue to answer a questionnaire that has already been completed, and who has access to these possibilities?
10. Who has access to publishing questionnaires?

**Questions for System B:**

1. Who has access to the system and how do they log on?
2. How is the roster made and updated, and who is responsible for it?
3. What possibilities are there in the system to look at rosters and who has access to the different rosters?
4. How is the second nurse (the one who does not initiate the swap) informed that another nurse wishes to swap duties?
5. How is the head of the ward informed about requested swaps?
6. How are the nurses (both the one who initiated the swap and the other) informed about whether a swap has been accepted?
7. Are there possibilities to delete a swap, that is, to return to the original roster?
8. What functionality exists in the system for transferring duties to other systems (for example, the system for paying out wages)?

*PAPER II:*

# Towards an Inspection Technique for Use Case Models

*Bente Anda and Dag I. K. Sjøberg*

SEKE'02 -14th IEEE Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 15-19, 2002, pp. 127-134.

## Abstract

A use case model describes the functional requirements of a software system and is used as input to several activities in a software development project. The quality of the use case model therefore has an important impact on the quality of the resulting software product. Software inspection is regarded as one of the most efficient methods for verifying software documents. There are inspection techniques for most documents produced in a software development project, but no comprehensive inspection technique exists for use case models. This paper presents a taxonomy of typical defects in use case models and proposes a checklist-based inspection technique for detecting such defects. This inspection technique was evaluated in two studies with undergraduate students as subjects. The results from the evaluations indicate that inspections are useful for detecting defects in use case models and motivate further studies to improve the proposed inspection technique.

**Keywords:** Use cases, Inspections

# 1 Introduction

In a use case driven software development process, a use case model is used as input to planning and estimating the software development project as well as to design and testing. A use case model may also be part of the contract between the customers and the developers regarding the functionality of a system. The quality of a use case model in terms of correct, complete, consistent and well understood functional requirements is thus important for the quality of the resulting software product.

Inspections [7] have proved to be an efficient means for detecting defects and improving quality in software documents. The structuring of the functional requirements in a use case model motivates an inspection technique with strategies for discovering defects adapted to this particular structure. The literature on use case models recommends reviews of the use case model to assure quality [3,10,16], and many organizations conduct such reviews with varying degree of formality. The increasing use of UML has motivated the development of a family of reading techniques for UML diagrams [18], but to the knowledge of the authors, no comprehensive inspection technique exists for use case models.

Several guidelines for constructing use case models exist. We conducted an experiment to evaluate the effects of two different sets of guidelines [2]. The results from that experiment show that the use of guidelines has an effect on the quality of the resulting use case models. This motivated a study to investigate how the quality of a use case model can be further improved through the use of an inspection technique.

Knowledge of typical defects is a prerequisite for developing and evaluating an inspection technique for use case models. Therefore, a taxonomy of defects in use case models and their consequences, was developed. The inspection technique is based on the taxonomy and on several recommendations for checklists found in the literature.

Any new technique should be evaluated, and the inspection technique was evaluated in a student project of a large undergraduate course in software engineering, and in a controlled experiment with 45 students as subjects.

The conducted studies indicate that inspections are useful for detecting defects in use case models, and suggest how the proposed inspection technique can be improved. Future work will focus on developing a basic technique that can be calibrated to suit a particular organization or application domain.

The remainder of this paper is organized as follows. Section 2 includes a definition of software inspections and describes different inspection techniques for requirements specifications. Section 3 presents a taxonomy of typical defects in use case models. Section 4 describes the proposed inspection technique. Section 5 reports the studies undertaken to evaluate the inspection technique. Section 6 concludes and suggests future work.

# 2 Software Inspections

This section describes the technique software inspection and the related techniques reviews and walkthroughs. Some particular inspection techniques for requirements specifications are also described.

## 2.1 Inspections, Reviews and Walkthroughs

An *inspection* is defined as a formal evaluation technique in which software requirements, design or code are examined in details by a person or group to detect defects, violations of development standards, and other problems [4]. The objective of an inspection is to:

- verify that the software element(s) satisfy its specifications,
- verify that the software element(s) conform to applicable standards,
- identify deviation from standards and specifications, and
- collect software engineering data (such as defect and effort data).

In addition to detecting defects in a software document and thus improving quality, inspecting a software document in a systematic manner teaches the developers to build better software [18].

Inspection techniques that use a non-systematic way of identifying defects are called *ad hoc techniques* [4]. The inspectors must utilize their own experience and knowledge to identify defects. The results of this technique depend solely on the abilities of the inspectors.

In *checklist-based techniques* the inspectors are provided with a list of general defect classes to check against. This kind of inspection technique is most common in industry [6], but the technique has some shortcomings that are described in [11].

A *review* is defined as a manual process that involves multiple readers checking a document for anomalies and omissions [19]. It is generally recommended that representatives of the different stakeholders in a project should participate in the review, but that they should look for different problems and defects.

A *walkthrough* is a peer group review of a software document [21]. It involves several people, each of whom plays a well defined role. A typical walkthrough involves at least one person, most often usually the author, whose job it is to introduce the document to the rest of the group.

In the context of requirements documents, inspections are recommended for defect detection, reviews for consensus and walkthroughs for training [8]. It is also recommended that an inspection should be performed before the two other activities to remove defects, which are noise in the process of achieving consensus on the requirements and in the walkthrough process.

### 2.2 Inspections of Requirements Specifications

The problems with ad hoc and checklist-based inspection techniques have been attempted remedied by introducing a scenario-based technique [14], where a checklist is used as a starting point for a more elaborate technique. The elements of the checklist were replaced by scenarios implementing the elements. The claims for the scenario-based technique is that it teaches the inspectors how to read the requirements documents in order to detect defects, and it offers a strategy for decomposition enabling each of the inspectors to concentrate on distinct parts of the requirements document. The scenario-based technique proved more effective than ad hoc and checklist-based inspections [14]. However, several replications of this evaluation have been conducted with varying results [12,13,15]. The replication reported in [15] found weak support for the original results, while the two other replications did not find the scenario technique superior to the two other techniques.

Different alternative decomposition strategies have been attempted to give the inspectors distinct responsibilities. One strategy is used in perspective-based reading. This technique is based on the classification of defects according to the perspectives represented by the different stakeholders in the project. The perspectives should be tailored to the needs of the various stakeholders, typical perspectives are clients or end-users, developers and testers. The reading technique for the user perspective involves constructing a use case model from the textual requirements.

Another strategy is used in the inspection technique usage-based reading, where a prioritized use case model is taken as input [20]. Its strength is claimed to be that it makes the inspectors focus on the defects that are important for the future users of the system.

## 3 Defects in Use Case Models

To develop and evaluate an inspection technique for use case models, we need knowledge of typical defects in use case models and of their consequences. Table 1 shows our proposal for a taxonomy of defects in use case models. The defects are divided into omissions, incorrect facts, inconsistencies, ambiguities and extraneous information [17]. In addition to the general quality issues presented in [2], we have considered different stakeholders to find a comprehensive list of defects.

Clients and end users want to be sure that they get the expected functionality. In terms of use case models this implies the following:

- The correct actors should be identified and described.
- The correct use cases should be identified and should describe how the use case goals [5] are reached. The actors should be associated with the correct use cases.
- The flow of events in each use case should be realistic in that it leads to the fulfilment of the use case goal. The use case descriptions should be easy to understand for users who are unfamiliar with use case modelling so that the

described functionality can be verified. This implies that the use cases should be described at an appropriate level of detail.
- The functionality should be correctly delimited through the use of pre- and post-conditions and variations.

Project managers need to plan software projects. For example, when estimating software projects, use case models can be used successfully [1]. To support the planning:
- the use case model should cover all the functional requirements, and
- all the interactions between the actor and the system that are relevant to the user, both in the normal flow of events and the variations should be described.

Designers will apply use case models to produce an object-oriented design. Therefore:

- the use of terminology should be consistent throughout the use case descriptions, and
- the use case descriptions should be described at a suitable level of detail. There should be no details of the user interface or internal details that put unnecessary constraints on the design

Testers will apply use case models to test that the functionality is correctly implemented. Therefore:
- the prerequisites (pre-conditions) for the execution of a use case, and the outcome (post-conditions) of each use case should be testable, and
- all terms in the use case descriptions should be testable.

A use case model consists of a diagram that gives an overview of the actors and the use cases, and of textual descriptions of each use case detailing out the requirements, typically using a template [5]. Use cases can, however, be described using many different formats [9]. The actual format may have an impact on the ease of detecting certain defects. For example, it should always be clear what are the pre- and post-conditions, of a use case. If a template format is used, pre- and post-conditions will usually be easily detectable. If the use cases are described with free text, on the other hand, it may be necessary to search the use case description for the information.

Some defects may also be specific to the format. To verify that applicable standards are followed, the inspection technique must be tailored to the actual format used. The proposed taxonomy is based on a format with normal flow of events and variations as well as the use case starting condition (trigger), and pre- and post-conditions. There are both simple and elaborate variants of the template format. We have chosen the simple template since our aim is to present a basic taxonomy that can be further extended to fit an actual project.

**Table 1. Taxonomy of defects in use case models**

| | **Actors** | **Use cases** | **Flow of events** | **Variations** | **Relation between use cases** | **Trigger, pre- and post-conditions** |
|---|---|---|---|---|---|---|
| **Omissions** | Human users or external entities that will interact with the system are not identified | Required functionality is not described in use cases. Actors have goals that do not have corresponding use cases | Input or output for use cases is not described. Events that are necessary for understanding the use cases are missing | Variations that may occur when attempting to achieve the goal of a use case are not specified | Common functionality is not separated out in included use cases | Trigger, pre- or post-conditions have been omitted |
| **Incorrect facts** | Incorrect description of actors or wrong connection between actor and use case | Incorrect description of a use case | Incorrect description of one or several events | Incorrect description of a variation | Not applicable | Incorrect assumptions or results have led to incorrect pre- or post-conditions |
| **Inconsist-encies** | Description of actor is inconsistent with its behaviour in use cases | Description is inconsistent with reaching the goal of the use case | Events that are inconsistent with reaching the goal of the use case they are part of | Variations that are inconsistent with the goal of the use case. | Inconsistencies between dia-gram and descriptions, inconsistent terminology, inconsistencies between use cases, or different level of granularity | Pre- or post-conditions are inconsistent with goal or flow of events |
| **Ambiguities** | Too broadly defined actors or ambiguous description of actor | Name of use case does not reflect the goal of the use case | Ambiguous description of events, perhaps because of too little detail | Ambiguous description of what leads to a particular variation | Not applicable | Ambiguous description of trigger, pre- or post-condition |
| **Extraneous information** | Actors that do not derive value from/provide value to the system | Use cases with functionality outside the scope of the system or use cases that duplicate functionality | Superfluous steps or too much detail in steps | Variations that are outside the scope of the system | Not applicable | Superfluous trigger, pre-or post-conditions |
| **Consequences** | Expected functionality is unavailable for some users or interface to other systems are missing | Expected functionality is unavailable | Too many or wrong constraints on the design or the goal is not reached for the actor | Wrong delimitation of functionality | Misunder-standings between different stake-holders, inefficient design and code | Difficult to test the system and bad navigability for users between different use cases |

**1. Actors**
- 1.1. Are there any actors that are not defined in the use case model, that is, will the system communicate with any other systems, hardware or human users that have not been described?
- 1.2. Are there any superfluous actors in the use case model, that is, human users or other systems that will not provide input to or receive output from the system?
- 1.3. Are all the actors clearly described, and do you agree with the descriptions?
- 1.4. Is it clear which actors are involved in which use cases, and can this be clearly seen from the use case diagram and textual descriptions? Are all the actors connected to the right use cases?

**2. The use cases**
- 2.1. Is there any missing functionality, that is, do the actors have goals that must be fulfilled, but that have not been described in use cases?
- 2.2. Are there any superfluous use cases, that is, use cases that are outside the boundary of the system, do not lead to the fulfilment of a goal for an actor or duplicate functionality described in other use cases?
- 2.3. Do all the use cases lead to the fulfilment of exactly one goal for an actor, and is it clear from the use case name what is the goal?
- 2.4. Are the descriptions of how the actor interacts with the system in the use cases consistent with the description of the actor?
- 2.5. Is it clear from the descriptions of the use cases how the goals are reached and do you agree with the descriptions?

**3. The description of each use case**
- 3.1. Is expected input and output correctly defined in each use case; is the output from the system defined for every input from the actor, both for normal flow of events and variations?
- 3.2. Does each event in the normal flow of events relate to the goal of its use case?
- 3.3. Is the flow of events described with concrete terms and measurable concepts and is it described at a suitable level of detail without details that restrict the user interface or the design of the system?
- 3.4. Are there any variants to the normal flow of events that have not been identified in the use cases, that is, are there any missing variations?
- 3.5. Are the triggers, starting conditions, for each use case described at the correct level of detail?
- 3.6. Are the pre- and post-conditions correctly described for all use cases, that is, are they described with the correct level of detail, do the pre- and post conditions match for each of the use cases and are they testable?

**4. Relation between the use cases:**
- 4.1. Do the use case diagram and the textual descriptions match?
- 4.2. Has the include-relation been used to factor out common behaviour?
- 4.3. Does the behaviour of a use case conflict with the behaviour of other use cases?
- 4.4. Are all the use cases described at the same level of detail?

**Fig. 1. Checklist for inspections of use case model**

## 4 An Inspection Technique for Use Case Models

The checklist approach was chosen as a starting point for developing an inspection technique for use case models, despite the problems mentioned in Section 2, because several such checklists already exist [3,10,16,22]. Checklists were also the starting point for more elaborate inspection techniques for other software documents as described in Section 2.2. In this paper, we have chosen the term inspection instead of the term review because our focus is on detecting defects rather than on reaching

consensus on the requirements. Based on the taxonomy in Section 3 and several recommendations for checklists for use cases models, we developed the checklist in Figure 1.

Our aim was a basic inspection technique which would be generally applicable. The checklists proposed in [3,10,16,22] contain some aspects that we have not included in our checklist because they were considered too specialized for our purpose and applicable only for some projects.

In [3] it is recommended to consider how a use case model fits with the overall business process model. For each use case it should be clear which business event initiates it, and which source it originates from.

The approach described in [10] differs from ours in that it recommends that a review should verify that the use cases meet technical criteria and that the user interfaces are consistent. They recommend that use case granularity should be verified. This is done by asking whether the use case model would be easier to understand if some use cases were split, and whether one path through a use case can be implemented in one iteration in the development project.

Separate reviews for completeness and for potential problems are recommended in [16]. The review for completeness should verify that the use cases fit the architecture and that the user interface matches the use cases. The review for potential problems should be conducted with clients or end users, and developers. Clients and end users should focus on whether they agree on the assumptions behind the functional requirements. Developers should focus on whether they have sufficient information to start construct the system. In addition to our checks, the checklist proposed in [22] recommends prioritization of the use cases for delivery and classification of their importance.

## 5 Evaluation of the Inspection Technique

To empirically evaluate the proposed inspection technique, two studies were conducted: Study 1 and Study 2. The aim of this evaluation was to investigate to what extent the inspection technique would improve defect detection[2].

### 5.1 Study 1

Study 1 was conducted over two semesters (autumn 2000 and autumn 2001) in the context of an undergraduate course in software engineering. The students were taught use case modelling in two lectures, and had exercises in seminars. The course also included a project where the students were organized in teams and developed a small software system.

The students in the course were in their 3rd or 4th year. A large number, approximately 40%, had part-time jobs as software developers or had previously worked with software development. About half of them were familiar with UML and

[2] The material used in the evaluation can be found at http://www.ifi.uio.no/forskning/grupper/isu/forskerbasen

use case modelling, mostly from previous courses; only a couple had applied use case modelling professionally.

### 5.1.1 Design of Study 1

In the project, the students were organized in teams of clients and developers. Two different systems were developed; each team was clients for one system and developers for the other system. In autumn 2000, 139 students divided into 31 teams either developed a hospital roster management system or a system for conducting opinion polls on the internet. In autumn 2001, 118 students divided into 27 teams either developed a hotel room allocation system or a sales management system. The client teams made informal, textual requirements specifications and handed those over to their developers. The developers then constructed use case models. The pairs of teams also had a couple of meetings to clarify the requirements.

During the autumn 2000, the client teams wrote an evaluation report on the use case models they had received. Very few defects were reported even though an analysis by the authors of this paper showed that the use case models did contain many defects.

The following year, autumn 2001, we wanted to investigate whether an inspection technique would improve the teams' ability to detect defects. We also wanted to examine whether the different perspectives represented by respectively the clients and the developers would lead to detection of different defects.

The development teams and the client teams conducted inspections using the checklist in Section 4. Each use case model was therefore inspected twice. The client teams were asked to focus on whether the use case model described the expected functionality. The development teams were asked to focus on whether there was enough information to create a good design, and later test that the delivered system was in accordance with the functional requirements. The teams had approximately two weeks available for this task. The teams registered effort spent on the inspections. There was a large difference in effort between the different teams, ranging from 2 to 30 hours, partly because of differences regarding how many of the team members participated in the inspections. Nevertheless, the registered hours showed that the teams were serious about the inspections.

The inspections resulted in reports that described the defects found. These reports were analyzed, and then the use case models were inspected by two people, one of them the first author of this paper. We decided to accept all the defects found by the teams as actual defects. Since the textual requirements specifications were different for all the teams, we considered the students' knowledge of the requirements to be better than ours. The defects were classified according to the categories described in Section 3.

### 5.1.2 Results from Study 1

Table 2 shows the total number of defects found by the client teams and the development teams distributed by the categories presented in Section 3. The number of defects found by both the client team and the development team are shown in the row marked *common*. The number 3 in the 'Actors' column means that out of the 92 defects concerning actors in the 27 use case models, only 3 were identified by both

the client team and the development team of a particular system. The defects found in the final inspection by the first author and one assistant, and not found by neither the client team nor the development team are shown in the row *not found*.

Almost all the teams found defects and suggested corrections. We consider these results as good indications that the checklists helped the teams to detect defects. This is further supported by the fact that we found very few defects that had been missed by the teams.

The results show that the clients found most defects, on average more than twice as many as the developers, and that there were strikingly few common defects. This indicates a large difference between what is considered a defect in a use case model.

**Table 2. Total number of defects detected in the student project**

| | Actors | Use cases | Flow of events | Variat-ions | Relation between use cases | Trigger, pre/post conditions |
|---|---|---|---|---|---|---|
| **Clients** | 60 | 49 | 59 | 37 | 8 | 48 |
| **Developers** | 26 | 17 | 29 | 24 | 3 | 42 |
| **Common** | 3 | 4 | 2 | 7 | 0 | 9 |
| **Not found** | 6 | 8 | 46 | 3 | 5 | 10 |
| **Total** | 92 | 74 | 134 | 64 | 16 | 100 |

The defects found by the clients frequently appeared to be due to expectations regarding functionality of the system that they had not expressed in the informal requirements specifications nor in the meetings with the development team, but which they missed when they read through and inspected the use case model. Many defects found by the developers were actually elements of the functionality that should have been described more precisely, but these weaknesses were not necessarily defects.

The difference in defects found by the clients and developers indicates that an inspection technique based on different perspectives, similar to perspective-based reading for textual requirements [17], may be useful for use case models. It also shows that after the inspection reviews of the use case models involving different stakeholders in the project can be useful in order to reach consensus on the requirements.

## 5.2 Study 2

Two weeks after the inspections were completed in the student project autumn 2001, a controlled experiment was conducted with 45 of the students as subjects. The students volunteered to participate in the experiment

### 5.2.1 Design of Study 2

The participants received a textual requirements specification for the hospital roster management system which had been implemented in the student project the previous year. The requirements for the system were based on the requirements for an actual system for a Norwegian hospital. These students were unfamiliar with that system. They received a use case model for the system with several defects inserted by us.

These defects were similar to the defects that we had detected when the system was used in the student project the previous year.

Half of the participants received a checklist similar to the one used in Study 1, shown in Section 4. The checklist in this experiment was slightly adapted to suit a context where the participants were unfamiliar with the actual use case model. Therefore, the checklist explicitly asked the participants to read the textual requirements specification and mark possible actors and their goals, that is, possible use cases. The other half was not given any particular inspection technique; they used ad hoc inspection.

The inspections were performed individually. The students made a list of all the defects, and they commented on the use case model when a defect was detected.

The duration of the experiment was three hours. The students were paid to participate. We did not want time to be a constraint on the experiment, so the subjects where given ample time. They were given an extra task after the inspection to keep them busy for three hours, but it was stressed that they did not have to complete the extra task.

The inspected use case models and the lists of defects were analyzed by the same two persons as in Study 1. The defects were classified according to the categories described in Section 3.

### 5.2.2 Results from Study 2

Table 3 shows that the inspectors who used the checklist found slightly more defects regarding the actors and the use cases than did those using the ad hoc technique. These defects are the most important, and could have had very serious consequences if not detected early in the development process. The inspectors using the ad hoc technique found more defects in the other categories, but overall the difference in the number of defects detected was negligible. However, Figure 2 shows that the difference in time spent on the inspection is significant in favour of the ad hoc approach. Therefore, using the checklist was more time-consuming without leading to more defects being found.

Table 3 further shows that all the subjects found quite a lot of the defects regarding actors, use cases, triggers and pre- or post-conditions. They did not find many of the defects in the flow of events or defects with superfluous or missing variations. This indicates that such errors are difficult to detect without having developed a more thorough understanding of the requirements.

Table 4 shows that the standard deviation was larger in most categories for those using the ad hoc approach, probably because the subjects using the ad hoc approach used more varied strategies for finding defects.

In addition to detecting defects that were deliberately planted in the use case model, most of the inspectors made some suggestions for how the requirements and the use case model could be improved. They also detected some “false” defects, that is, they were not really defects. There was no noticeable difference between the two inspection approaches.

The results indicate that a checklist or a specific inspection technique may not be particularly useful when the inspectors already have good knowledge about the defects they are expected to find as had the inspectors in this case; they had recently performed similar inspections. On the contrary, experienced inspectors may be more

efficient without a checklist. This supports previous work that did not show any particular differences between ad hoc, checklists or scenario-based techniques [4,12,13]. A checklist may, however, be a good means to assure that a task is performed seriously.

**Table 3. Average number of defects found in the experiment**

| | Actors | Use cases | Flow of events | Variat-ions | Relation between use cases | Trigger, pre/post conditions |
|---|---|---|---|---|---|---|
| **Checklist** | 3,0 | 2,0 | 1,0 | 0,6 | 0,4 | 3,7 |
| **Ad hoc** | 2,8 | 1,8 | 1,7 | 1,0 | 0,6 | 4,6 |
| **Actual defects** | 4 | 4 | 5 | 6 | 4 | 10 |

**Table 4. Standard deviation for number of defects found in the experiment**

| | Actors | Use cases | Flow of events | Variat-ions | Relation between use cases | Trigger, pre/post conditions |
|---|---|---|---|---|---|---|
| **Checklist** | 0,8 | 1,1 | 1,0 | 0,7 | 0,6 | 2,4 |
| **Ad hoc** | 1,0 | 1,2 | 1,0 | 1,0 | 0,5 | 3,0 |
| **Actual defects** | 4 | 4 | 5 | 6 | 4 | 10 |

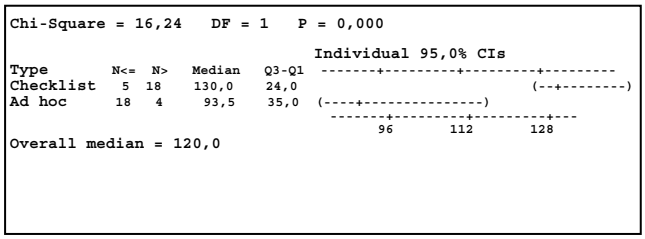


**Fig. 2. Moods Median test on time spent**

## 5.3 Threats to Validity

The taxonomy of defects in use case models presented in Section 3 requires more work to be more complete. A different taxonomy may lead to different results for the proposed inspection technique. There were some defects in the use case models that were difficult to assign to a specific category. Therefore, the distribution of defects in the different categories might have been slightly different if the defects had been categorized differently.

Both evaluations were conducted with undergraduate students on use case models of rather small scale. We may get different results if evaluations are performed with inspectors who have more experience with use case modelling and inspections. A follow-up experiment with professional software developers is therefore planned.

The size and format of the use case models may have impacted the results. Larger use case models could have made it infeasible for the inspectors to inspect the whole use case model. The experiment reported in [15] shows that the format of the textual requirements may have a larger impact on the inspectors' ability to detect defects than does the inspection technique. However, the template style used in these evaluations is frequently recommended and is a commonly used format [5].

Study 1 shows that the client teams found most defects. These teams may not be representative of typical clients as they were also developers and thus familiar with use case modelling.

Study 2 shows that the checklist-based inspection technique was not more efficient than the ad hoc technique. In this study, the participants were familiar with the checklist and the classes of defects from the student project. The students performing the ad hoc inspections may therefore have used elements of the checklist even though they did not have the checklist available when performing the inspection.

## 6 Conclusions and Future Work

The quality of a systems' use case model is important for the quality of the resulting software product. In this paper we introduced a tentative taxonomy of defects in use case models and a checklist-based inspection technique to detect such defects. The checklist was evaluated in a student project and subsequently in a controlled experiment, also with students.

We presented anecdotal evidence that inspections may be a useful means to improve the quality of use case models because the teams using the checklist in the student project found many more defects than did the teams not using such a checklist. Clients and developers in our studies found very different defects even though they used the same inspection technique. This indicates that different stakeholders should participate in the inspection.

The controlled experiment showed that experienced inspectors were more efficient without using the checklist. Therefore, more work is needed to establish appropriate inspection techniques. The following activities are planned:

- Studies of use case reviews in actual software development projects to investigate how different stakeholders search for and detect defects in use case models.
- Refinement of the taxonomy and the inspection technique. We plan to investigate how the questions can be tailored to the needs of different stakeholders. We also intend to study how the questions best can be phrased in order to provide appropriate strategies for detecting defects in use cases described with different formats.

## Acknowledgements

We thank Kirsten Ribu for help with the analysis. We also thank all the students who took part in the evaluation of the use case inspection technique.

## References


1. Anda, B., Dreiem, H., Sjøberg, D.I.K., and Jørgensen, M. Estimating Software Development Effort Based on Use Cases - Experiences from Industry. UML'2001, Toronto, Canada, October 1-5, 2001, LNCS 2185 Springer-Verlag, pp. 487-502.
2. Anda, B., Sjøberg, D.I.K. and Jørgensen, M. Quality and Understandability in Use Case Models. ECOOP'2001, June 18-22, 2001, LNCS 2072 Springer-Verlag, pp. 402-428.
3. Armour, F. and Miller, G. Advanced Use Case Modelling. Addison-Wesley, 2000.
4. Cheng, B. and Jeffery, R. Comparing Inspection Strategies for Software Requirement Specifications. Proceedings Australian Software Engineering Conference. IEEE Comput. Soc, Los Alamitos, CA, USA, 1996.
5. Cockburn, A. Writing Effective Use Cases. Addison-Wesley, 2000.
6. El Emam, K. and Laitenberger, O. Evaluating Capture-Recapture Models with Two Inspectors. IEEE Trans. on Softw. Eng., Vol. 27(9), pp. 851-864, September 2001.
7. Fagan, M.E. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, Vol. 15(3), pp. 182-211, 1976.
8. Gilb, T. and Graham, D. Software Inspection. Addison-Wesley, 1993.
9. Hurlbut, R.R. A Survey of Approaches for Describing and Formalizing Use Cases. Technical Report: XPT-TR-97-03, Expertech, Ltd., 1997.
10. Kulak, D. and Guiney, E. Use Cases: Requirements in Context. Addison-Wesley, 2000.
11. Laitenberger, O., Atkinson, C., Schlich, M. and El Emam, K. An experimental comparison of reading techniques for defect detection in UML design documents. Journal of Systems and Software, Vol. 53(2), pp. 183-204, August 2000.
12. Lanubile.F. and Visaggio, G. Assessing defect detection methods for software requirements inspections through external replication, ISERN-96-01, January 1996.
13. Miller, J., Wood, M., Roper, M. and Brooks, A. Further Experiences with Scenarios and Checklists. Empirical Software Engineering, Vol. 3(1), pp. 37-64, January 1998.
14. Porter, A.A., Votta, L.G. and Basili, V.R. Comparing Detection Methods for Software Requirements Inspections: a Replicated Experiment. IEEE Trans. on Softw. Eng., Vol. 21(6), pp. 563-575, June 1995.
15. Sandahl, K, Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M. and Ohlsson, N. An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections, Empirical Software Engineering, Vol. 3(4), pp. 327-354, December 1998.
16. Schneider, G. and Winters, J. Applying Use Cases – A Practical Guide. Addison-Wesley, 1998.
17. Shull, F., Rus, I. and Basili, V. How Perspective-Based Reading Can Improve Requirements Inspections. IEEE Computer, Vol. 33(7), pp. 73-79, July 2000.
18. Shull, F., Travassos, G.H., Carver, J. and Basili, V.R. Evolving a Set of Techniques for OO Inspections. CS-TR-4070 and UMIACS-TR-, October 1999.
19. Sommerville, I. Software Engineering, 5th Ed. Addison-Wesley, 1996.
20. Thelin, T., Runeson, P. And Wohlin, C. An Experimental Comparison of Usage-Based and Checklist-Based Reading. WISE 2001.
21. Yourdon, E. Structured Walkthroughs. Prentice-Hall, 1989.
22. www.mcbreen.ab.ca/papers/QAUseCases.htm

*PAPER III:*

# Understanding Use Case Models

*Bente Anda and Magne Jørgensen*

Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Workshop, 5 June, 2000 at 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, pp. 94-102, 2000.

## Abstract

Use Case Modeling is a technique for handling the functional requirements in a software development project. The Use Case Model can serve as a means of communication between the different stakeholders in a project. It is used in planning the project and is updated and used during the project. In order to reduce the possibilities for misunderstandings and differences in understanding, it would be useful to be able to evaluate to what extent the different stakeholders have understood the model and also to detect differences in interpretation. Low comprehension or differences in interpretation may indicate a need for more effort on specifying the requirements. If this is not feasible, it may be necessary to assume a higher risk when planning and estimating the project. We propose using knowledge on how humans understand text from cognitive psychology in the design of an experiment with a twofold goal: Investigate methods for measuring comprehension of Use Case Models and analyze the differences in understanding.

**Keywords:** Understanding, Requirements engineering, Use Case Models, Schema Theory

# 1 Introduction

Requirements engineering is a critical part of the software development process. According to a survey carried out by the Standish Group[3] incomplete requirements are the most important factor in explaining why software projects fail. In addition, problems related to requirements engineering are one cause of many of the other factors that the Standish Group found to be important, for example, changing requirements and specifications as described in [1]. Model clashes are another possible factor in failed software projects [2]. Different stakeholders in a software development project will often have different priorities. These priorities are reflected in different success models of the system. When the models conflict, the project encounters difficulties and may fail.

An increasing number of organizations attempt to improve their requirements engineering process by adopting a particular technique called Use Case Modeling introduced by Ivar Jacobson [3]. The Use Case Model of a system should capture all the functional requirements of the system to be developed and provide a means of communication between the different stakeholders in a development project, i.e. clients, potential users of the system, managers and different groups of developers. Several notations exist [3-6] for Use Case Modelling. The notations differ with respect to the content of the templates that are used for describing the Use Cases and with respect to the degree of formalism. Independently of the notation used, different levels of abstraction are possible, so the requirements engineering team must make decide how much detail they want to put into the specification, both in terms of eliciting and describing the requirements.

There is currently no detailed account of the cognitive processes involved in checking software requirements. While there are several studies on how programmers understand programs [7-14], we have not been able to find any studies on human understanding of Use Case Models. This lack of studies on Use Case Model understanding means that the guidelines and practices on how Use Cases should be described to ease their understandability and provide a basis for a common understanding of the requirements, is highly subjective. A common understanding of the requirements may also reduce the possibility of a model clash. We intend to investigate how understandability of Use Case Models can be measured and how different stakeholders understand Use Case Models. We have recently started projects in two different organisations aimed at investigating this issue

How people understand texts and construct meaning from them, have been studied for several years [15-17]. We therefore believe that knowledge about how humans understand text can provide important insight into how they understand representations of software systems in the form of Use Case Models.

To summarize, the major challenge raised in this paper is:

[3] http://www.standishgroup.com/chaos.html. A study based on a survey of US companies.

*Are there findings and methods from the research on how humans construct meaning from text that can be used successfully when studying how different stakeholders read and understand requirements specifications in the form of Use Case Models?*
*This challenge includes: How can experiences from design of experiments on how humans construct meaning from text be reused in experiments on how requirements specifications in the form of Use Case Models are understood?*

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 gives a brief description of how humans understand text. Finally section 4 sketches an experiment for evaluating how different stakeholders understand Use Case Models.

## 2 Related work

Work has been done on evaluating the understandability of the syntax and semantics of languages for requirements specifications [18]. This work resulted in a set of guidelines for improving understandability of specification languages, applicable to designers of such languages and to specification developers.

Perspective-Based Reading is a technique for detecting defects in requirements specifications [19-21]. The different reviewers of the requirements documents assume different perspectives when inspecting the documents. Several studies have been conducted in order to investigate the effects of this technique compared with checklists or "ad hoc" reading. The results from these studies are not conclusive, and this work differs from ours in that we are primarily interested in understanding, as we believe that to be a prerequisite for detecting errors and handling changes in the requirements.

A set of guidelines for authoring Use Cases have been developed by the ESPRIT 21.903 CREWS research project. These guidelines are intended to improve completeness and correctness and to avoid ambiguities. They were evaluated in [22]. The guidelines give some advice on level of abstraction, but they were not evaluated regarding whether they improve understandability of a Use Case Model.

Much work has been done on program understanding which is similar to understanding a Use Case Model because both include understanding a text expressed with a particular notation. In both cases it is necessary with knowledge of the application domain, and of the syntax and semantics of the language used. There are differences due to programs being expressed in a formal language, while Use Cases are expressed in everyday language, only guided by a template. In addition, programs will only be understood by programmers, while people with a much more varied background should understand requirements. Research on program understanding indicates differences in understanding due to differences in knowledge of the domain, knowledge of the programming language and general problem solving strategy.

There are several cognitive theories potentially useful for describing understanding and complexity. They may be useful to explain why some models are difficult to understand for people, depending on context and background. Examples of such theories are:

- "Chunking theory". A "Chunk" is a unit of knowledge that has become familiarized and has a place in the memory's "index". This theory has, for example, been used to explain performance differences between expert and novice programmers in understanding a piece of program [23].
- "Acceptability principles theory" [24]. According to this theory, difficulties in understanding are caused by the violation of some "acceptability principles", i.e. commonly accepted rules for how things are to be understood. For example, in [25] the experienced programmers did only outperform the novices when the programs they had to change followed "accepted rules for programming". If, for example, the program had functions and variable names that did not correspond with the content or function, there was no significant difference between the experienced programmers and the novices.

In addition to these theories there are a lot of other theories and findings within cognitive science and psychology that may be useful for our study. In this paper, however, we have attempted to use Schema Theory [15], a theory of how humans understand text, and we investigate how this theory can be applied to the understanding of Use Case Models.

## 3 Schema Theory applied to the understanding of Use Case Models

Schema Theory is a frequently used theory in studies on how humans understand text.

Schema Theory was introduced by Bartlett in 1932 [15]. He discovered how people's understanding and remembrance were shaped by their expectations. He suggested that these expectations could be modeled using schemata.

A schema is defined in [16] as an abstract knowledge structure in the sense that it summarizes what is known about a variety of cases that differ in many particulars.

Schemata are used to make inferences while we read, and inference drawing is a prerequisite for comprehension. The readers create a mental model of the situation and events referred to in the text. A cognitive schema is invoked when we are confronted with a small element of the schema. The schema will then have an important influence on how the rest is understood. Schemata are built from past experience.

A simple example is a schema for "Buying a book on the Internet". If we imagine two stakeholders that are involved in the development of a bookshop on the Internet, the stakeholders may have different schemata for "buying a book on the Internet". One of the stakeholders may have acquired his schema only from buying books in "real" bookshops. The other stakeholder may have experience as a bookseller and a schema acquired from that experience.

The first stakeholder's schema may include the following:

1. Place order
2. Pay
3. Receive order

The second stakeholder's schema may include:

1. Receive information about books
2. Register as customer
3. Place order
4. Receive information about alternative methods of payment
5. Select method of payment
6. Pay
7. Receive order

Due to different experiences in the past, different stakeholders will have developed different schemata and may consequently interpret the Use Cases Model differently.

The effects of different stakeholders using different schemata may result in the following:

- Terms will have different meanings for different people
- Elements are left out because it is believed that they are so obvious that it is not necessary to explicitly include them
- Misunderstandings

Detienne [9] used Schema Theory in the design of an experiment aimed at analyzing the cognitive mechanisms involved in program understanding. She found that the programmers used schemata related to domain knowledge together with schemata related to programming. She found that differences relating to which schema where used in a situation depended on the programmer's experience.

Pressley and McCormick present a number of experiments where people have read stories and found different inconsistencies or have remembered different things according to which schema they used [17]. They also referred to experiments that showed that the participants accepted sentences to be in the text when they were not, but fitted their schema.

We believe that Schema Theory is potentially useful in a study of the understandability of Use Case Models. Schema theory shows that the reader's schemata may be as important as the text itself in determining how a text is understood. The theory stresses the importance of understanding which models the readers use in order to be able to increase their understanding of what is read. These results are important input to our study. Schema theory also allows us to reuse experimental design from other fields, and we will have the possibility to compare our results with existing work. In addition to Schema theory we intend to identify and evaluate other theories and findings within cognitive science and psychology which may be useful for our study.

A Use Case Model consists of a graphical overview, a Use Case Map, and narratives describing each Use Case. We will examine the understandability of both parts in our experiment. Literature on diagram comprehension [26-27] has been searched for theory or methods that could be used in investigating how Use Case Maps are understood. We consider this work less relevant in our context because Use Case Maps use very few symbols and has a very limited syntax compared with most diagram techniques.

## 4 Experiment

We plan to carry out an experiment with the following two goals:

- The primary goal is to evaluate different experimental designs that can be used in the design of a method for detecting differences in the understanding of a Use Case Model.
- The secondary goal is more exploratory, it is to detect differences in understanding of a Use Case Model, differences in the schemata that are used and possible reasons for these differences, and to get indications regarding how notation and level of detail in the Use Case Model influence understanding.

This section describes the first version of an experimental design. The design is inspired by experiments on understanding text referred to in Bartlett [15] and Pressley and McCormick [17] and by experiments on program understanding performed by Von Mayrhauser and Lang [7] and Detienne [9]. The design of the experiments on Perspective-Based Reading [20] will also be used to further elaborate this design.

Hypothesis
- It is possible to find an optimal level of detail for the Use Case Model for a specific purpose and under given circumstances.

Subjects
The subjects will be different stakeholders in a software development project.

There will be approximately 20 subjects. The subjects will have varying experience and knowledge about the application domain, and varying experience and training in Use Case Modeling.

Artifacts
The subjects will be presented with Use Case Models describing the functional requirements of two small applications. The following variations of Use Case Models will be used.

- Use Case Maps with highlevel Use Case templates.
- Use Case Maps with relations between Use Cases and expanded Use Case templates with scenarios.

The first variant represents a typical description of a Use Case Model developed very early in the project, while the second variant is more elaborate.

Method of investigation
The primary goal of the experiment is to find a method that can be used to assess comprehension and detect misunderstandings and differences in interpretation. Several methods will be used and evaluated. A combination of methods may be used for each subject. The methods are:

- *Think-aloud protocol [7]*. The subjects are encouraged to say what they are thinking when reading and understanding the Use Case Model.

- *Questions [9].* After reading the Use Case Model the subjects respond to questions. The questions should indicate comprehension and ability to use information extracted from the Use Case Model.
- *Reproduction [15].* The subjects will be asked to describe the Use Case Model in their own words after having read it.
- *Inconsistencies and lack of precision in the Use Case Model [17].* Deliberate inconsistencies and lack of precision will be introduced in the Use Case Model. The experiment will investigate, through questions after the subjects have read the Use Case Model, whether there are differences in the inferences made when information is lacking, and if there are differences in how many errors/inconsistencies are found by the different stakeholders.

For all methods, the goal is to detect differences in the schemata, the search strategies and differences regarding which inferences are made.

Threats to validity

The following may represent threats to the validity of the results:

- Few subjects mean that they will study the Use Case Models individually. This situation may differ from how they usually read requirements documents. The limited number of subjects may also make it difficult to obtain statistical validity.
- Each subject will investigate several Use Case Models. There may therefore be learning during the experiment, which may affect the result.
- The Use Case Models used in the experiment will describe the functional requirements of two small applications. These Use Case Models may not be representative of other, larger applications.

## Acknowledgements

We gratefully acknowledge the support from our industrial partner, Icon Medialab, in particular Alexander Woxen and Per Einar Dybvik. The research project is funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry).

## References

1. Pfleeger, S. L. *Software Engineering. Theory and Practice*. Prentice Hall Inc., 1998.
2. Boehm, B. & Port, D. "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them". Software Engineering Notes, vol. 24, no.1, 36-48, 1999.
3. Jacobson, I. et al. *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley, 1992.
4. Constantine, L. L. & Lockwood, L. A. D. *Software for Use. A Practical Guide to the Models and Methods for Usage-Centered Design*. Addison-Wesley, 1999.

5. Regnell, B., Andersson, M. & Bergstrand, J. "A Hierarchical Use Case Model with Graphical Representation". Proceedings of Second IEEE International Symposium on Requirements Engineering (RE'95), York, UK, 1995.
6. Cockburn, A. ”Structuring Use Cases with goals”. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, Ha.T.TR.95.1, http://members.aol.com/acockburn/papers/usecases.htm, 1995.
7. Von Mayrhauser, A & Lang, S. "A Coding Scheme to Support Systematic Analysis of Software Comprehension". IEEE Transactions on software Engineering, vol. 25, no. 4, 526-540, 1999.
8. Brooks, R. "Towards a Theory for the Comprehension of Computer Programs". International Journal of Human-Computer Studies, vol. 51, nr. 2, 197-211, 1999.
9. Detienne, F. "Une application de la théorie des schémas a la comprehension de programmes". Le Travail Humain, tome 51, no. 4, 335-350, 1988.
10. Adelson, B. “Problem solving and the development of abstract categories in programming languages”. Memory and Cognition, vol. 9, no. 4, 422-433, 1981.
11. Miara, R. et al. “Program indentation and comprehensibility”. Communications of the ACM, vol. 26, 861-867, 1983.
12. Gilmore, D. J. and Green, T.R.G. “Comprehension and recall of miniature programs”. Journal of Man-Machine Studies, vol. 21, 31-48, 1984.
13. Letovsky, S. and Soloway, E. “Delocalized plans and program comprehension”. IEEE Software, vol. 3, no. 4, 41-49, 1986.
14. Linos, P. et al. “Facilitating comprehension of C-programs: an experimental study”. Proceedings IEEE Second Workshop on Program Comprehension, 55-63, 1993.
15. Bartlett, F. C. *Remebering. A Study in Experimental and Social Psychology*. Cambridge University Press, 1932.
16. Anderson, R.C., & Pearson, P.D. *A schema theoretic view of basic processes in reading*. P.D. Pearson (Ed.) Handbook on reading research. New York, Longman, 1984.
17. Pressley, M. & McCormick, C. B. *Advanced Educational Psychology for Educators, Researchers and Policymakers*. Harper Collins, 1995.
18. Britton, C. & Jones, S. "The untrained eye: how languages for software specification support understanding in untrained users". Human-Computer-Interaction, vol. 14, nr.1-2, 191-244, 1999.
19. Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. & Zelkowitz, M." The Empirical Investigation of Perspective-Based Reading". Empirical Software Engineering Journal, vol 1, nr. 2, 133-146, 1996.
20. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. & Zelkowitz, M. *Lab Package for the Empirical Investigation of Perspective.Based Reading*, 1998. Available at: http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html.
21. Regnell, B., Runeson, P. & Thelin T. "Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements", Technical Report CODEN: LUTEDX(TETS-7172)/1-40/1999, Dept. of Communication Systems, Lund University, 1999.
22. Ben Achour, C., Rolland, C., Maiden, N.A.M. & Souveyet, C. "Guiding Use Case Authoring: Results of an Empirical Study". Proceedings IEEE Symposium on Requirements Engineering, IEEE Comput. Soc, Los Alamitos, CA, USA,1999.
23. Schneiderman & Mayer. “Syntactic/semantic interactions in programmer behaviour: a model and experimental results”. International Journal of Computer and Information Sciences, vol. 8, 219-238, 1979.
24. Kosslyn , S. M. “Understanding charts and graphs”. Applied Cognitive Psychology, 3, 185-223, 1989.

25. E. Soloway, B. Adelson, and K. Ehrlich, “Knowledge and process in the comprehension of computer programs,” in *The nature of expertice*, M. T. H. Chi et al., Eds.: Lawrence Erlbaum Assoc.,129-152, 1988.
26. Winn, W. "An Account of How Readers Search for Information in Diagrams". Contemporary Educational Psychology, vol. 18, nr. 2, 162 - 185, 1993.
27. Glasgow, J., Narayanan, N.H. & Chandrasekaran, B. *Diagrammatic reasoning : cognitive and computational perspectives*. Menlo Park, Calif.: AAAI Press, 1995.

***PAPER IV:***

# Estimating Software Development Effort based on Use Cases – Experiences from Industry

*Bente Anda, Hege Dreiem, Dag I.K. Sjøberg and Magne Jørgensen*

M. Gogolla and C. Kobryn (Eds.): UML 2001 - 4th International Conference on the Unified Modeling Language, Toronto, Canada, October 1-5, 2001, pp. 487-502, LNCS 2185, Springer-Verlag.

## Abstract.

Use case models are used in object-oriented analysis for capturing and describing the functional requirements of a system. Several methods for estimating software development effort are based on attributes of a use case model. This paper reports the results of three industrial case studies on the application of a method for effort estimation based on *use case points*. The aim of this paper is to provide guidance for other organizations that want to improve their estimation process applying use cases. Our results support existing claims that use cases can be used successfully in estimating software development effort. The results indicate that the guidance provided by the use case points method can support expert knowledge in the estimation process. Our experience is also that the design of the use case models has a strong impact on the estimates.

**Keywords** Use cases, estimation, industrial experience

# 1 Introduction

Use case modelling is a popular and widely used technique for capturing and describing the functional requirements of a software system. The designers of UML recommend that developers follow a use case driven development process where the use case model is used as input to design, and as a basis for verification, validation and other forms of testing [8].

A use case model defines the functional scope of the system to be developed. The functional scope subsequently serves as a basis for top-down estimates[4]. A method for using use case models as a basis for estimating software development effort was introduced by Karner [13]. This method is influenced by the function points method and is based on analogous use case points. The use of an adapted version of the use case points method is reported in [3] where it was found that attributes of a use case model are reliable indicators of the size of the resulting functionality. Use case models have also been found well suited as a basis for the estimation and planning of projects in a software improvement project [16]. However, we have been unable to find studies that describe the use case points estimation process in details. This paper describes a pilot study on three system development projects. The aim of this paper is to provide a detailed description of the method used and experiences from applying it.

Our study was conducted in a software development company located in Norway, Sweden and Finland. The company has a total of 350 employees; 180 are located in Norway. Its primary areas of business are solutions for e-commerce and call-centers, in particular within banking and finance. The company uses UML and RUP in most of their software development projects, but currently there is neither tool nor methodological support in place to help the estimation process. The company wishes to improve the process of estimating software development effort. This is the origin of the process improvement initiative reported in this paper.

We compared estimates based on use case points for three development projects with estimates obtained by experts, in this case senior members of the development projects, and actual effort. Our results support findings reported elsewhere [3,13,16] in that use case models may be suitable as a basis for effort estimation models. In addition to supporting other studies, we have experienced that the guidance provided by the use case points method appears to reduce the need for expert knowledge in the estimation process.

UML does not go into details about how the use case model should be structured nor how each use case should be documented [17]. Therefore, use case models can be structured and documented in several alternative ways [19]. An experiment described in [2] indicated that the understandability of a use case model is influenced by its structure, and our results show that the structure of the use case model has a strong

[4] In general, a top-down estimate is produced applying an estimation method to factors believed to influence the effort necessary to implement a system. The estimation method gives the total software development effort, which may then be divided on the different activities in the project according to a given formula. Adding up expected effort for all the activities planned in a project, on the contrary, produces a bottom-up estimate.

impact on the precision of the estimates. In particular, we experienced that the following aspects of the structure had an impact:

- the use of generalization between actors[5]
- the use of included and extending use cases[6]
- the level of detail in the use case descriptions

An important prerequisite for applying a use case based estimation method is that the use cases of the system under construction have been identified at a suitable level of detail. The use case model may be structured with a varying number of actors and use cases. These numbers will affect the estimates. The division of the functional requirements into use cases is, however, outside the scope of this paper.

The remainder of this paper is organized as follows. Section 2 gives an overview of the use case points method. Section 3 describes related work and presents alternative methods and tools for estimation based on use cases. Section 4 describes the three development projects that were used as case studies and how data was collected from them. Our results are reported in Section 5. Lessons learned are reported in Section 6. Section 7 discusses threats to the validity of our results. Section 8 concludes and suggests directions for future work.

## 2 The Use Case Points Method

This section gives a brief overview of the steps in the use case points method as described in [18]. This estimation method requires that it should be possible to count the number of transactions in each use case. A transaction is an event occurring between an actor and the system, the event being performed entirely or not at all.[7] The four steps of the use case points method are as follows:

1. The actors in the use case model are categorized as *simple*, *average* or *complex*. A simple actor represents another system with a defined API; an average actor is another system interacting through a protocol such as TCP/IP; and a complex actor may be a person interacting through a graphical user interface or a web-page. A weighting factor is assigned to each actor category:
   - Simple: Weighting factor 1
   - Average: Weighting factor 2
   - Complex: Weighting factor 3

[5] Two actors can be generalized into a *superactor* if there is a large description that is common between those two actors.

[6] Common behaviour is factored out in included use cases. Optional sequences of events are separated out in extending use cases [17].

[7] Appendix A shows a use case from one of the development projects used in this study. The basic flow of events in the use case consists of 7 transactions. The use case is documented according to a template used throughout the company. The template ressembles those recommended in [6].

The total *unadjusted actor weight* (*UAW*) is calculated counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the products.

2. The use cases are also categorized as *simple*, *average* or *complex*, depending on the number of transactions, including the transactions in alternative flows. Included or extending use cases are not considered. A simple use case has 3 or fewer transactions; an average use case has 4 to 7 transactions; and a complex use case has more than 7 transactions. A weighting factor is assigned to each use case category:

   - Simple: Weighting factor 5
   - Average: Weighting factor 10
   - Complex: Weighting factor 15

   The *unadjusted use case weights (UUCW)* is calculated counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The UAW is added to the UUCW to get the *unadjusted use case points* (*UUPC*).

3. The use case points are adjusted based on the values assigned to a number of technical factors (Table 1) and environmental factors (Table 2).

Each factor is assigned a value between 0 and 5 depending on its assumed influence on the project. A rating of 0 means the factor is irrelevant for this project; 5 means it is essential.

*The Technical Factor (TCF)* is calculated multiplying the value of each factor (T1 –T13) in Table 1 by its weight and then adding all these numbers to get the sum called the *TFactor*. Finally, the following formula is applied:

$$TCF = 0.6 + (.01*TFactor)$$

*The Environmental Factor (EF)* is calculated accordingly by multiplying the value of each factor (F1 – F8) in Table 2 by its weight and adding all the products to get the sum called the *Efactor*. The formula below is applied:

$$EF = 1.4+(-0.03*EFactor)$$

The *adjusted use case points (UCP*) are calculated as follows:

$$UCP = UUCP*TCF*EF$$

**Table 1. Technical complexity factors**

| Factor | Description | Wght |
|---|---|---|
| T1 | Distributed system | 2 |
| T2 | Response or throughput performance objectives | 2 |
| T3 | End-user efficiency | 1 |
| T4 | Complex internal processing | 1 |
| T5 | Reusable code | 1 |
| T6 | Easy to install | 0.5 |
| T7 | Easy to use | 0.5 |
| T8 | Portable | 2 |
| T9 | Easy to change | 1 |
| T10 | Concurrent | 1 |
| T11 | Includes security features | 1 |
| T12 | Provides access for third parties | 1 |
| T13 | Special user training facilities are required | 1 |

**Table 2. Environmental factors**

| Factor | Description | Wght |
|---|---|---|
| F1 | Familiar with Rational Unified Process | 1.5 |
| F2 | Application experience | 0.5 |
| F3 | Object-oriented experience | 1 |
| F4 | Lead analyst capability | 0.5 |
| F5 | Motivation | 1 |
| F6 | Stable requirements | 2 |
| F7 | Part-time workers | -1 |
| F8 | Difficult programming language | -1 |

4. Karner [13] proposed a factor of 20 staff hours per use case point for a project estimate, while Sparks states that field experience has shown that effort can range from 15 to 30 hours per use case point [21]. Schneider and Winters recommend that the environmental factors should determine the number of staff hours per use case point [18]. The number of factors in F1 through F6 that are below 3 are counted and added to the number of factors in F7 through F8 that are above 3. If the total is 2 or less, use 20 staff hours per UCP; if the total is 3 or 4, use 28 staff hours per UCP. If the number exceeds 4, they recommend that changes should be made to the project so the number can be adjusted. Another possibility is to increase the number of staff hours to 36 per use case point.

# 3 Related Work

This section reports two experiences with estimation based on use case points. Two alternative methods and one tool for estimation based on use cases are described. Finally, use case points are compared to function points.

## 3.1 Reported Experiences with Estimation Based on Use Cases

Arnold and Pedross reported experiences from using use case points to measure the size of 23 large-scale software systems [3]. Their method for counting use case points was inspired by, but not identical to, Karner's method. Their experience was that the use case points method is a reliable indicator of the size of the delivered functionality. However, they observed that the analyzed use case models differed much in the degree of details and believed that the measured size may have differed according to this degree. They also found that free textual use case descriptions were insufficient to measure the software size.

Martinsen and Groven reported a software process improvement experiment aimed at improving the estimation process using a use case model in estimating a pilot project [16]. Before the improvement project, the requirements specification was only loosely coupled with the effort and cost estimates. The requirement specification was written in natural language, which was found too informal to be a good basis for the necessary revision of the cost estimate or for restricting the implementation within the cost estimate. Adopting use case modelling, the customer and developers had a common, documented understanding of the requirements. The pilot project experienced an overrun on the estimates, but the overrun was smaller than the average for previous, similar projects. Hence, they found use cases useful as a basis for estimation and planning.

## 3.2 Methods and Tools for Use Case Estimation

Alternative methods for estimation based on use cases are described in [7] and [20]. In [7] the use case model is a basis for counting function points, which in turn may be used to obtain an estimate of effort. In [20] the use case model is used to estimate the number of lines of code (LOC) in the finished system. This number of LOC is subsequently used as the basis for an estimate.

These two methods appear more complex than the one we have used as they respectively make assumptions on the relationship between use cases and function points, and between use cases and the number of LOC in the finished system. These assumptions have not been tested. The advantage of these methods, however, is that they may exploit the extensive experience with estimation using function points or lines of code.

Optimize [22] is a tool that provides estimates based on use case models. Optimize measures the size of the problem counting and classifying scope elements in a project. The set of use cases in the project's use case model is one kind of scope element. Other possibilities are, for example, the project's classes, components and web-pages. Qualifiers are applied to each scope element. The complexity qualifier defines each scope element as simple or complex. The tool provides a set of default metrics, extrapolated from experience on more than 100 projects. The user can also customize metric data to produce estimates calibrated for an organization. Optimize organizes the scope elements and metric data to compute an estimate of effort and cost. We intend to evaluate this tool more thoroughly. So, far we have only tried it briefly on data from one of the development projects. Our impression is that the tool requires calibration to the particular organization to provide a reasonable estimate. Moreover, the cost of purchase and training makes it less accessible than the method with associated spreadsheet that we have used.

## 3.3 Use Case Points and Function Points

The number of function points measures the size of a software application in terms of its user required functionality [1]. Although the calculation of use case points has been strongly influenced by function points, there are several important differences leading to different strengths and weaknesses:

- The function point standards do not require that the input documents follow a particular notation. Use case points are based on the use case model. This means that it is easier to develop estimation tools that automatically count use case points; the counting is based on available documents (use case models). This is an important difference, since counting function points frequently requires much effort and skill.
- There are international standards on how to count function points. The concept of use case points, on the other hand, has not yet reached the level of standardization. Without a standard describing the appropriate level of detail in the requirement description, i.e., the use case model, there may be very large differences in how different individuals and organizations count use case points. Hence, it may currently be difficult to compare use case point values between companies. As reported in [9;14], even with a counting standard there may be significant differences in how people count function points.

## 4 Data Collection

Table 3 shows some characteristics of the three software development projects used in our case studies.

**Table 3. Characteristics of three software development projects**

| Characteristic | Project A | Project B | Project C |
|---|---|---|---|
| Size | 7 months elapsed time, 4000 staff hours | 3 months elapsed time, 3000 staff hours | 4 months elapsed time, 3000 staff hours |
| Software architecture | Three-tier, established before the project | Three-tier, known, but not established in advance | As project B |
| Programming environment | Java (Visual Café and JBuilder), Web Logic | MS Visual Studio | Java (Jbuilder), Web Logic |
| Project members | 6 developers with 0 to 17 years experience | 6 developers with 0 to 12 years experience | 5 developers with 2 to 10 years experience, 4 consultants were involved part time. |
| Application domain | Finance | CRM (Customer relationship manage-ment within banking), part of a larger solution | Banking (support for sale of credit cards) |

Our research project was conducted in parallel with project A during a period of seven months. Projects B and C, on the other hand, were finished before the start of our research. We collected information about the requirements engineering process and about how the expert estimates were produced. We also collected information about the use case models and actual development effort.

Data from project A was collected from the project documents, i.e., the use case model, iteration plan and spreadsheets with estimates and effort, and from several interviews with project members. Data from project B was collected from project documents, and from e-mail communication with people who had participated in the project. In this project the available documentation consisted of a detailed requirements specification with several use case diagrams and textual descriptions of use cases, project plan and time sheets recording the hours worked on the project. Data from project C was collected from project documents, including a requirements specification with brief textual descriptions of each use case, a use case model in Rational Rose with sequence diagrams for each use case, project plan and initial estimates, and from an interview with two of the project members. The collected data is shown in Table 4.

**Table 4. Data collection in the three development projects**

| Data element | Project A | Project B | Project C |
|---|---|---|---|
| Requirements engineering | 600 hours spent on requirements specification. Relatively stable throughout the project. | Effort not available. Some serious changes in the requirements during the project. | Effort not available. Stable requirements throughout the project. |
| Expert estimate | Produced by a senior developer with 17 years experience. The esti-mation process was influenced by the function points method; effort was estimated per screen. | Produced by a senior developer with 7 years experience. | Produced by three developers with between 6 months and 9 years experience. |
| The use case model | No included or extending use cases. Example of a use case in Appendix A. The customer reviewed the use case model and read through the use cases. | Included many small use cases (containing only 1 or 2 transactions). Contained many included and extending use cases as and a large number of actors. | Contained many included and extending use cases. Each use case was described by a brief textual description and a sequence diagram. |
| The use case estimation process | A senior member of the project team counted and assessed actors and use cases and assigned values to the technical and environmental factors. Values were inserted into a spread-sheet to produce an estimate. The estimation process took approxi-mately one hour when the use case model was completed and well understood by the person performing the estimation. | The senior developer who had produced the initial expert estimate counted and assessed actors and use cases and assigned values to the technical and environ-mental factors. An alternative estimate was produced by the first author counting and assessing actors and use cases based on the textual requirements documents. A spread-sheet was used in the estimation.[1] | The project manager assigned values to the technical and environ-mental factors and also assessed the complexity of each actor. The first author counted use cases from the requirements document and a Rational Rose Model and assessed their complexity. A spread-sheet was used to produce an estimate. |
| Time sheets | Actual effort was computed from time sheets. The time sheets were structured to enable registering effort on each use case. | Hours were recorded according to some predefined activities. Actual effort was calcu-lated adding up all the activities in the project. | As project B |

[1]Two different estimates were produced due to different interpretation on how to count actors and usecases.

# 5 Results

The results are shown in Table 5. Despite of no customisation of the method to this particular company, the use case estimates are fairly close to the estimates produced by the experts.

**Table 5. Expert estimate, use case estimate and effort (in hours)**

| Project | Expert estimate | Use case estimate | | Actual effort |
|---|---|---|---|---|
| A | 2730 | 2550 | | 3670 |
| B | 2340 | 3320 | 2730 | 2860 |
| C | 2100 | 2080 | | 2740 |

In projects A and C, the use case estimate ended up only slightly below the expert estimate but a bit below the actual effort. The use case estimate for project B is close to actual effort and somewhat higher than the expert estimate.

The first use case based estimate for project B (3320) was produced by the authors with information about actors and use cases given by the senior developer in the project. This estimate was very much higher than the original expert estimate, and it was also higher than the actual effort. We believe that this is because trivial actors were counted, such as printer and fax, and also included and extending use cases. We therefore decided to calculate a second estimate where actors and use cases were counted from the use case model. The actors that provided input to the system or received output from it were generalized into two superactors. Only those two were counted, not the individual actors. This reduced the number of actors from 13 to 6. The included and extending use cases were omitted. We used the same technical and environmental factors in the second estimate as in the first estimate. This resulted in an estimate of 2730 hours, which is very close to the actual effort on the project (2860 hours).

One reason why the use case estimate for project C ended up a bit below the actual effort may be that the project manager assigned too high values to the environmental factors regarding experience and capabilities of the team. For example, he assigned higher values than did the project manager of project B even though the two projects were conducted with similar teams regarding size and experience with software development. Using the same environmental factors as for project B, project C would get a use case estimate of 2597 hours which is very much closer to the actual effort.

The use case estimates for projects B and C were made after the completion of the projects. It was therefore easier to assess values for the technical and environmental factors than in a normal situation because the choice of values could be based on experience with the actual project. This indicates that the technical and environmental factors in the method are appropriate for this company, although there may be a need for some, but not extensive adjustments.

We consider these results promising for the use case points method. The expert estimates were produced by very competent senior developers with good knowledge of both the technology and the application domain. The results were obtained without any particular calibraton of the method, so it is likely that the use case estimates can be improved. Independent of the method used for estimation, we must expect

inaccuracies. Boehm states that these inaccuracies range up to 60 percent or more during the requirements phase [4]. The use case estimate for project A, the project with the largest difference, is 30 percent below actual effort.

Table 5 indicates a relationship between the use case estimate for a project and the effort needed to implement it. Based on this, we would expect a relationship between the size of each use case, measured in number of transactions and the actual effort on implementing the use case. The result of investigating this relationship is shown in Table 6.

**Table 6. Size and effort for each use case in project A**

| Use case | Number of transactions | Use case points | Deve-loper | Iter-ation | Expert estimate | Actual effort |
|---|---|---|---|---|---|---|
| 1. Fetch application | 16 | 15 | A | 0/1 | 42 h | 224 h |
| 2. Simulate application | 22 | 15 | B | 1/2 | 64 h | 301 h |
| 3. Automatic scoring | 11 | 15 | C | 1/2 | 86 h | 267 h |
| 4. Change application | 13 | 15 | C | 2 | 124 h | 144 h |
| 5. Assess credit-worthiness | 31 | 15 | | 2 | 170 h | |
| 6. Produce documents | 7 | 10 | B + D | 1/3 | 152 h | 122 h |
| 7. Register new application | 14 | 15 | All | 2/3 | 936 h | 647 h |
| 8. Notification of application | 5 | 10 | | 3 | 132 h | |
| 9. Transfer application to new responsible | 9 | 15 | | 3 | 82 h | |

For each use case, Table 6 shows the number of transcations, the number of use case points, the developer (anonymized), in which iteration the use case was developed, the expert estimate and the actual effort. The system was developed in four iterations, in the first iteration (iteration 0) the architecture was established and in the subsequent iterations the system was constructed. The realization of the use cases was divided on these iterations.

The functionality described in use cases 5, 7, 8 and 9 was realized as one unit. The total corresponding effort was registered on use case 7. Hence, six cells in the table are empty.

The use cases contain between 5 and 31 transactions. The number of use case points for each use case is calculated according to the description in Section 2. Use cases 1 through 4 were implemented by a single developer; use case 6 was implemented by two developers; and all the developers participated on use case 7. For most of the use cases, the basic flow was implemented in one iteration. Those use cases were then completed by the implementation of alternative flows in a later iteration. Originally, the expert had estimated effort per screen, but the screens were associated with use cases, so he managed to re-organize the estimates in order to show expected effort per use case. Actual effort shows effort on analysis, design and implementation for each use case and was calculated from the time sheets.

The sum of effort registered on the use cases is smaller than the total effort registered on the project because much of the effort was registered on activities that were not related to a particular use case.

Table 6 shows no relationship between the size of each use case measured as the number of transcations and the effort necessary to implement it. Possible reasons are:

- Estimates based only on the number of use cases, or on a division into simple and complex use cases, are equally precise to counting all the transactions. One way of investigating this further is to compare the size of each use case with the number of classes or lines of code necessary to implement it.
- Many factors influenced the registered effort for each use case, for example:
  - There were different levels of experience among the members of the project.
  - The use cases implemented in the last iterations could reuse design and code.
  - The structure of the time sheets was new to the project members, and they did experience some difficulties in registering effort exactly as intended.

With relatively few data, we are unable to correct for these confounding factors. Therefore, we need more data to further investigate the relationship between the size of a use case and the required effort to implement it.

# 6 Lessons Learned

This section presents a number of lessons learned from applying the use case points method.

## 6.1 The Impact of the Structure of a Use Case Model

When we applied the use case points method to the three projects, we experienced that the following aspects of the structure of a use case model had an impact on the estimates:

- The use of generalization between actors. The number of actors in a use case model affects the estimate. Our experience is that if the descriptions of two or more actors have a lot in common, the precision of the estimate is increased by generalizing the actors into a superactor and hence counting the actors only once.
- The use of included and extending use cases. Karner recommends that included and extending use cases should not be counted. Omitting such use cases for project B resulted in an estimate that was closer to the expert estimate and to the actual effort than if they were included. However, in project C we found it necessary to count the included and extended use cases as very much of the essential functionality was described using these constructs. In our opinion there is definitely a need to investigate this further. Separating out functionality in included and extending use cases reduces the number of transactions in the use cases from which the functionality is separated out, hence the estimate is reduced, but the functionality will still have to be implemented. Optional functionality can be

described either in an extending use case or in an alternative flow of events. This complicates the estimation process, because choosing an extending use case will result in a lower estimate than if an alternative flow is chosen.

- The level of details in the use case descriptions. The size of each use case is measured as the number of transactions. We experienced the following difficulties when counting transactions for each use case.
  - Almost all the use cases in project A were classified as complex. We believe that this indicates that the classification of complexity of the use cases should also include an alternative for very complex use cases, for example use cases with 15 or more transactions.
  - It is a challenge to decide the appropriate level of detail in each transaction when structuring a use case, as there are no metrics established to determine correct level of detail for all projects [15]. The level of detail in each transaction will affect the number of transactions, which subsequently has an impact on the estimate obtained with the use case points method.

## 6.2 Assigning Values to Technical and Environmental Factors

We experienced difficulties when trying to assign values to the technical and environmental factors because we lacked a basis for comparison. In some cases we had to guess what was meant by each factor and we had to try to recapture other projects with which this project could be compared. A particular problem here is that the environmental factors require an evaluation of the competency of the project team. People often have difficulties being neutral when they are asked to evaluate their own work. This may lead to problems if the project members themselves perform the use case estimation and have to assign values to the environmental factors. With more experience from using the method, it will be possible to reuse experiences from earlier projects and calibrate the method to fit the organization. However, this will require that use case models for different projects are structured with a consistent level of detail.

The choice of productivity rate for each use case point may also need calibration. We used a productivity rate of 20 staff hours pr. use case point, but we believe that this choice will depend on whether some activities are estimated outside the use case point method or not.

## 6.3 Time Sheets

We believe that the entities used in the estimation should correspond to the structure of the time sheets to enable feedback on the precision of the estimates to gradually improve the estimation process. In project A, the time sheets were organized as a table with five columns:

Activity, Use case, Functionality, Name of developer, Hours

Examples of activities are analysis and design, implementation and administration. There was, however, effort in the project that the developers found difficult to register on one particular use case, for example effort related to establishing the database and the client framework. It was some disagreement as to whether this effort should be registered on the first use case that was implemented or as a separate activity not related to a use case. The developers decided to do the latter.

We included all the activities performed after the use case model was completed in the actual effort. Some of the activities were untypical; that is, they would usually not be included in the organization's software development projects. Examples are effort on upgrading to a new version of a development tool and training in the development method and tools for new project members. We decided to include untypical activities also, because we believe that every project is special somehow. In our opinion, if the application of the use case model shall result in a complete top-down estimate, it must produce an estimate with some surplus time for unexpected activities.

### 6.4 Using Use Case Estimates

In our opinion, use case estimates should not substitute expert estimates. It seems sensible to combine models and human judgement [5]. We therefore believe that use case estimates can be used successfully in conjunction with expert estimates.

For estimators with little experience, the use case points method gives good support for estimation. Estimators may also find it useful to compare the unadjusted use case points for a system with unadjusted use case points from previous projects. Expert estimators may be strongly influenced by, for example, previous estimates or what the estimators believe will be the price to win [11;12], which in turn may result in large deviations from actual effort. The use of function points together with expert estimates has reduced such large deviations [10]. Use case points may be used to obtain the same effect. We also believe that the customers may more easily accept estimates if they know that an established method have been used to produce them.

## 7 Threats to Validity

In project A, the use case estimate and the expert estimate were produced in parallel. Hence, some of the information used in the expert estimate may have been reused in the use case estimate because of communication between the project members involved in producing the two estimates. We do not know whether this influence had any impact on the estimates.

Project B and C were finished before the start of the research project, so the actual effort was known when the use case estimates were produced. However, the project members who provided information to the estimation method were unable to make use of the information about the actual effort because they did not know the formula behind the use case estimate. Therefore, we believe that the actual effort had no impact on the use case based estimate.

In Project C there were no detailed textual descriptions of the use cases so the number of transactions in each use case was counted from sequence diagrams. Sequence diagrams typically describe the functionality at a lower level of detail than the textual use cases so there will usually not be equally many functions in a sequence diagram as there are transactions in the corresponding use case description. This means that the use case estimate for project C might have been slightly different if it had been produced from use case descriptions instead. After considering the actual sequence diagrams, we do not believe that this had a serious effect on the estimate in this case.

# 8 Conclusions and Future Work

We conducted three case studies on applying a method for estimating software development effort based on use cases, the use case points method. The results indicate that this method can be used successfully since the use case estimates were close to the expert estimates in our three case studies. In one case it was also very close to the actual effort. It is therefore our impression that the method may support expert knowledge. We intend to further study the precision of the use case point method compared with expert estimates. In our three projects the experts had much experience from similar projects. We will therefore conduct a study where the estimators have different levels of experience.

Moreover, our experience is that applying the use case point method in practice is not straightforward. For example, the choice of structure for the use case model has an impact on the estimates. There is consequently a need for further studies on the precision of the estimates when using the use case points method in different types of projects.

We also believe that it would be useful to investigate how the use case points method, which provides top-down estimates based on a measure of size, can be combined with other methods that provide bottom-up estimates. The purpose of using the estimation method investigated in this paper is to provide a complete estimate for all the activities in the project. Nevertheless, we believe that some of the activities in a development project do not depend on size or use case points, for example, training and establishing a new programming environment. Therefore, such activities should be estimated in alternative ways and then be added to the use case estimate to provide a final estimate.

Another direction we intend to pursue is comparing the different methods for use case estimation described in Section 3 with regards to precision of the estimates and the effort needed to produce them. The use case points method requires use cases to be described at a level of detail where each transaction is specified. This is not always the case in practice. We therefore believe it is useful to investigate whether other methods for use case estimation are suitable for use case models with less detail. The way use case models are described in a company should guide the choice of method for use case based estimation, or vice versa, a specific method for use case based estimation should guide the way the use case models are described in the company.

## Acknowledgements

We gratefully acknowledge the support from our industrial partner, Mogul, in particular Jon Ola Hove, Trond Andersen, Anne Hurlen, Skule Johansen, Helge Aarstein and Sigurd Stendal. The reported work was funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry).

## References

1. Albrecht, A.J. Measuring Application Development Productivity. Proceedings of Joint SHARE, GUIDE, and IBM Application Development Symposium. 1979.
2. Anda, B., Sjøberg, D., and Jørgensen, M. Quality and Understandability in Use Case Models. In Proc. 13th European Conference on Object-Oriented Programming (ECOOP'2001), Jørgen Lindskov Knudsen (editor), June 18-22 2001, Budapest, Hungary, LNCS 2072, Springer Verlag, pp. 402-428.
3. Arnold, P. and Pedross, P. Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department. Forging New Links. IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 490-493. 1998.
4. Boehm, B.W. *Software Engineering Economics*. Prentice-Hall. 1981.
5. Blattberg, R.C. and Hoch, S.J. Database models and managerial intuition: 50% model + 50% manager, *Management Science,* Vol. 36, No. 8, pp. 887-899. 1990.
6. Cockburn, A. Writing Effective Use Cases. Addison-Wesley. 2000.
7. Fetcke. T., Abran, A. and Nguyen, T.-H. Mapping the OO-Jacobson Approach into Function Point Analysis. International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-23). IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 192-202. 1998.
8. Jacobson, I., Christersson, M., Jonsson, P. and Övergaard, G. *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley. 1992.
9. Jeffery, D.R., Low, G.C. and Barnes, M. A comparison of function point counting techniques. *IEEE Transactions on Software Engineering*. Vol. 19, No. 5, pp. 529-532. 1993.
10. Jørgensen, M. An empirical evaluation of the MkII FPA estimation model, Norwegian Informatics Conference, Voss, Norway. 1997.
11. Jørgensen, M., Kirkebøen, G., Sjøberg, D., Anda and B., Bratthall, L. Human judgement in effort estimation of software projects, Beg, Borrow, or Steal Workshop, International Conference on Software Engineering, Limerick, Ireland. 2000.
12. Jørgensen, M. and Sjøberg, D.I.K. Software Process Improvement and Human Judgement Heuristics, *Accepted for publication in: Scandinavian Journal of Information Systems.* 2001.
13. Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21. December 1993.
14. Kemerer, F.K. Reliability of Function Points Measurement. *Communications of the ACM*. Vol. 36, No. 2, pp. 85-97. February 1993.
15. Kulak, D. and Guiney, E. *Use Cases: Requirements in Context*. Addison-Wesley. 2000.
16. Martinsen, S.A. and Groven, A-K. Improving Estimation and Requirements Management Experiences from a very small Norwegian Enterprise. Improvement in Practice: Reviewing Experience, Previewing Future Trends. The European Conference on Software Process Improvement (SPI 98). Meeting Management, Farnham, UK. 1998.

17. OMG Unified Modeling Language Specification, Version 1.3. June 1999. (http://www.rational.com/media/uml/post.pdf).
18. Schneider, G. and Winters, J. *Applying Use Cases – A Practical Guide*. Addison-Wesley. 1998.
19. Sendall, S. and Stroheimer, A. From Use Cases to System Operation Specification. Third International Conference on the Unified Modeling Language (UML'2000), York, UK. LNCS 1939; Springer Verlag, pp. 1-15. 2000.
20. Smith, J. The Estimation of Effort Based on Use Cases. Rational Software, White paper. 1999.
21. Sparks, S. and Kaspcynski, K. The Art of Sizing Projects, Sun World. 1999. (http://www.sunworld.com/sunworldonline/swol-12-1999/swol-12-itarchitect.html).
22. The Object Factory. Estimating Software Projects using ObjectMetrix, White paper. April 2000.

# Appendix A

## Use Case Specification : Transfer loan application to new responsible

### 1. Use Case Name: Transfer loan application to new responsible

#### 1.1 Brief description

The use case describes how a person responsible for a loan application can transfer it to another responsible.

### 2. Flow of Events

#### 2.1 Basic Flow

1. The responsible notifies the system that he wants to transfer a specific loan application to another responsible.
2. The system displays the name of the applicant and the reference number of the application.
3. The responsible verifies that the application is correct based on the name of the applicant and the reference number.
4. The system presents a list of groups of responsibles and users within each group.
5. The responsible may choose one group of responsibles and possibly one particular responsible.
6. The responsible requests the loan application to be transferred to the chosen (group of) responsible(s).
7. The system transfers the application to the chosen (group of) responsible(s).

The use case ends successfully.

#### 2.2 Alternative Flows

2.2.1 The responsible cancels the transfer

The responsible can cancel the transfer at any time and the use case ends.

2.2.2 Additional notification by mail

After the 5th step:

5.1 The responsible indicates that the new responsible should receive an e-mail telling him that he has received a new application for consideration.

5.2 The system automatically produces an e-mail message to the new responsible.

The use case resumes at step 6.

### 3. Special Requirements

### 4. Pre-Conditions

#### 4.1 The responsible must be logged on to the system

The system must be started and the responsible must be logged on correctly.

### 5. Post-Conditions

#### 5.1 The application has a valid status after saving

The application should be saved in the database with a valid status and in a consistent state.

#### 5.2 The application is assigned to one responsible or to a group of responsibles

The application should be assigned to one responsible or to a group of responsibles after the transfer is completed.

### 6. Extension Points

*PAPER V:*

# Comparing Effort Estimates Based on Use Case Points with Expert Estimates

*Bente Anda*

EASE 2002 - Empirical Assessment in Software Engineering, Keele, UK, April 8-10, 2002.

## Abstract.

Use case models are used in object-oriented analysis for capturing and describing the functional requirements of a system. Attributes of a use case model may therefore serve as measures of the size and complexity of the functionality of a system. Many organizations use a system's use case model in the estimation process.

This paper reports the results from a study conducted to evaluate a method for estimating software development effort based on use cases, the *use case points method*, by comparing it with expert[8] estimates. A system was described by a brief problem statement and a detailed use case model. The use case points method gave an estimate that was closer to the actual effort spent on implementing the system than most estimates made by 37 experienced professional software developers divided into 11 groups (MRE of 0.21 versus MMRE of 0.37).

The results support existing claims that the use case points method may be used successfully in estimating software development effort. They also show that the combination of expert estimates and method based estimates may be particularly beneficial when the estimators lack specific experience with the application domain and the technology to be used.

**Keywords.** Use Cases, Estimation

[8] The term expert is used in this paper to denote experienced estimators.

# 1 Introduction

Use case modelling is a popular and widely used technique for capturing and describing the functional requirements of a software system. A use case model has two parts, the use case diagram and the use case descriptions. The diagram provides an overview of actors and use cases. The use case descriptions detail the requirements. An actor represents a role that the user can play with regard to the system, and a use case represents an interaction between an actor and the system.

The *use case points method* is a method for estimating software development effort based on use cases. This method has shown large potential in two case studies reported in [3,12], but apart from these two studies, the method has not, to the author's knowledge, been subject to evaluation.

In the case study reported in [3], the use case points method was compared with expert estimates and actual effort for three industrial development projects. These projects lasted from 3 to 7 months with a total effort of 3000 to 4000 person hours. The systems constructed were for e-commerce and call-centres within banking and finance. The use case points method gave estimates that were almost as close to actual effort as the estimates produced by very experienced software developers with good knowledge of the application domains and the technology to be used. The use case based estimates were between 4.5% and 30% lower than the actual effort for the projects.

A modified version of the use case points method produced very precise estimates for two industrial development projects with a total effort of 10000 and 14000 person hours, respectively [12]. The first system was an internet application for a bank aimed at facilitating communication with customers. The other system was a real-time application that was part of a large commercial solution.

Those case studies motivated a new study, described in this paper, to investigate how the use case points method perform compared with experts, that is experienced software developers and estimators, with less experience with the application domain and the technology to be used

The overall motivation is that many organizations use a system's use case model in the estimation process. However, to the knowledge of the author, there are no standards for use case based estimation. In [15] it is recommended that method based estimates should be used to improve expert estimates, the results in [5] show that it is sensible to combine methods and human judgement.

This study was conducted as part of three courses on use case modelling in a large international IT company. Two of the courses were held in Denmark, one in Norway. The participants were very experienced software developers and project managers from Denmark, Norway and Sweden. All together there were 37 participants in the three courses, divided into a total of 11 groups. The groups were asked to estimate effort necessary for implementing a software system. A brief problem statement and a detailed use case model described the system. The participants had read through the problem statement and had worked on the use case model for almost one day before estimating the effort. The use case points method was also used to estimate the same

software system. The estimates were compared with the actual effort used to implement the system.

The estimation method gave an estimate with Magnitude of Relative Error (MRE) equal to 0.21 which was closer to the actual effort for the project than most of the estimates suggested by the groups of professional developers. The Mean Magnitude of Relative Error (MMRE) for the estimates made by the groups of experts was 0.37.

The results from this study therefore support existing claims that the use case points method may be used successfully in estimating software development effort. They also show that a combination of expert estimates and method based estimates may be more accurate than expert estimates alone, in particular when the estimators lack specific experience with the application domain and the technology to be used.

The remainder of this paper is organized as follows. Section 2 describes the use case points method. Section 3 describes the study in details. Section 4 presents the results. Section 5 discusses the design of the study. Section 6 concludes and describes how we the work reported in this paper is continued.

## 2 The Use Case Points Method

The use case points method was initially developed by Gustav Karner [9]. It is based on the function points method [2], and the aim was to provide a simple estimation method adapted to object-oriented projects. This section gives a brief overview of the method as described in [13]. The method requires that it should be possible to count the number of transactions in each use case. A transaction is an event occurring between an actor and the system, the event being performed entirely or not at all.[9] The four steps in the use case points method are as follows:

1. The actors in the use case model are categorized as *simple*, *average* or *complex*. A simple actor represents another system with a defined API; an average actor is another system interacting through a protocol such as TCP/IP; and a complex actor may be a person interacting through a graphical user interface or a web-page. A weighting factor is assigned to each actor category:

   - Simple: Weighting factor 1
   - Average: Weighting factor 2
   - Complex: Weighting factor 3

   The total *unadjusted actor weight* (*UAW*) is calculated by counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the products.

[9] Appendix A shows a use case from one of the development projects used in this study. The basic flow of events in the use case consists of 6 transactions. The use case is documented according to a template used throughout the company. The template resembles those recommended in [Cockburn, 2000].

2. The use cases are also categorized as *simple*, *average* or *complex*, depending on the number of transactions, including the transactions in alternative flows. Included or extending use cases are not considered. A simple use case has 3 or fewer transactions; an average use case has 4 to 7 transactions; and a complex use case has more than 7 transactions. A weighting factor is assigned to each use case category:

   - Simple: Weighting factor 5
   - Average: Weighting factor 10
   - Complex: Weighting factor 15

   The *unadjusted use case weights (UUCW)* is calculated by counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The UAW is added to the UUCW to get the *unadjusted use case points* (*UUPC*).

3. The use case points are adjusted based on the values assigned to a number of technical factors (Table 1) and environmental factors (Table 2). These factors are meant to account for effort that is not related to the size of the task.

**Table 1. Technical complexity factors**

| Factor | Description | Wght |
|---|---|---|
| T1 | Distributed system | 2 |
| T2 | Response or throughput performance objectives | 2 |
| T3 | End-user efficiency | 1 |
| T4 | Complex internal processing | 1 |
| T5 | Reusable code | 1 |
| T6 | Easy to install | 0.5 |
| T7 | Easy to use | 0.5 |
| T8 | Portable | 2 |
| T9 | Easy to change | 1 |
| T10 | Concurrent | 1 |
| T11 | Includes Security features | 1 |
| T12 | Provides access for third parties | 1 |
| T13 | Special user training facilities are required | 1 |

**Table 2. Environmental factors**

| Factor | Description | Wght |
|---|---|---|
| F1 | Familiar with Rational Unified Process | 1.5 |
| F2 | Application experience | 0.5 |
| F3 | Object-oriented experience | 1 |
| F4 | Lead analyst capability | 0.5 |
| F5 | Motivation | 1 |
| F6 | Stable requirements | 2 |
| F7 | Part-time workers | -1 |
| F8 | Difficult programming language | -1 |

Each factor is assigned a value between 0 and 5 depending on its assumed influence on the project. A rating of 0 means that the factor is irrelevant for this project; 5 means that it is essential.

*The TechnicalComplexity Factor (TCF)* is calculated by multiplying the value of each factor in Table 1 by its weight and then adding all these numbers to get the sum called the *TFactor*. Finally, the following formula is applied:

$$TCF = 0.6 + (.01*TFactor)$$

*The Environmental Factor (EF)* is calculated accordingly by multiplying the value of each factor in Table 2 by its weight and adding all the products to get the sum called the *Efactor*. The formula below is applied:

$$EF = 1.4+(-0.03*EFactor)$$

The *adjusted use case points (UCP*) are calculated as follows:

$$UCP = UUCP*TCF*EF$$

4. Karner proposed a factor of 20 person hours per use case point for a project estimate. Schneider and Winters recommend that the environmental factors should determine the number of person hours per use case point [13]. The number of factors in F1 through F6 that are below 3 are counted and added to the number of factors in F7 through F8 that are above 3. If the total is 2 or less, use 20 person hours per UCP; if the total is 3 or 4, use 28 person hours per UCP. If the number exceeds 4, they recommend that changes should be made to the project so the number can be adjusted. Another possibility is to increase the number of person hours to 36 per use case point.

A spreadsheet is used to implement the method and produce an estimate. The method provides an estimate in total number of person hours.

The use case points method can be criticised from a theoretical point of view as has been the function points method. The addition and subsequent multiplication of ordinal values is, for example, theoretically invalid [10]. However, the function points method has been shown to predict effort reasonably well for many types of systems, and the aim of this study is to extend the empirical evaluation of the use case points method.

There are several other methods for use case based estimation. The methods differ in that size and complexity of the use cases are measured differently, and they also consider different technical and environmental factors. The company in which the study was conducted has its own method for use case based estimation. A commercial tool, *Optimize* [16,18], provides estimates when a use case model for the project is available.

Two alternative methods for estimation based on use cases are described in [7,14]. The method described in [7] maps attributes of the use case model into function points. In [14] a certain number of lines of code is assumed for each use case, and the

total number of lines of code is used as a basis for the estimate. A metric suite for use case models, which can be used for estimation, is suggested in [11], but a complete estimation method is not presented.

The use case points method was chosen for this study because it had already showed good results in two previous case studies [3,12]. The company specific method is confidential and is therefore not described here.

## 3 The Study

This section describes the participants taking part in the study, the context, the procedure of the study and the material used.

### 3.1 Participants

The study was conducted as part of three courses on use case modelling in a large international IT company. The course schedule was the same in all the three courses. The participants were experienced developers, business analysts and project managers. The author was one of the instructors on the last two courses. All the participants had extensive experience of requirements engineering, and they had experience from estimating their own work. None were previously familiar with estimation based on use cases, but some were familiar with the function points method. Table 3 gives some characteristics of the participants in the three courses.

**Table 3**. **Description of three courses on use case modelling**

| Characteristic | Course 1 | Course 2 | Course 3 |
|---|---|---|---|
| No. of participants | 11 | 14 | 12 |
| Nationality of participants | Danish, Norwegian and Swedish | Danish | Norwegian |
| No. of groups | 3 | 4 | 4 |
| Average experience with software development | 10 years | 7 years | 16 years |
| Experience with use case modelling | All had attended previous courses. Some had professional experience. | Most had attended previous courses. Some had professional experience. | Some had attended previous courses. None had professional experience. |
| Experience with similar projects | Some. | Some. Mostly experience from large batch-systems. | Few. Mostly experience from mainframe and maintenance. |

## 3.2 Context

The duration of the courses was two days. The course schedule consisted of presentations on different aspects of use case modelling, one small assignment on the first day and a larger assignment on the second day. At the end of the first day, the participants were given a problem statement and a system context diagram for a system to be constructed. They were asked to read this through to be prepared for the assignment. On the second day, the participants were randomly divided into groups. The groups worked on constructing a use case model for most of the day. At the end of this assignment, the groups were given a complete solution consisting of 5 actors and 22 use cases. They were then given a 30 minutes lecture on estimation in general and on how attributes of a use case model can be used as a measure of the size of the functionality. The use case points method was not presented in this lecture.

## 3.3 Procedure of the Estimation Task

Based on the use case model with 4 actors and 22 use cases that had been handed out, each group spent 15 minutes on estimating team size, elapsed project time in months and total effort in person months. Each group was asked to estimate the effort that they would have used themselves.

In all the groups the members discussed among themselves to reach a result. Some of the estimates were activity based, that is, made by adding up the expected effort for the activities in the development project more or less independently of the use cases. Other estimates were produced by the groups attempting variants of use case based estimation ranging from simply counting all the use cases and assuming one person months for each, to classifying the use cases into easy, medium and complex and suggesting a number of weeks effort for each category of use case.

All the groups had an estimate ready after 15 minutes as they were used to following a tight course schedule for almost two days.

## 3.4 Material

This section gives a brief description of the problem statement that the groups used as a basis for constructing a use case model, the use case model that was handed out before the estimation exercise, and the project that implemented the actual system. Since the course material was taken from an actual project, the company does not permit the presentation of the complete problem statement and use case model.

### 3.4.1 Problem statement

The description was as follows:
The system will be the IT part of a service for shopping through the Internet and the delivery of products to the customers' homes. The service should be available through a call-centre or the Internet. The system will consist of an order taking facility for both the server and the clients. A list of functional and non-functional requirements to the system was included.

Examples of functional requirements:

1. Register new customer
2. Search for product
3. Make an order

Examples of non-functional requirements:

1. Customer details, especially credit card information, should be protected
2. The solution should be scaleable to handle potentially large increases in use
3. Product details should be kept up-to-date

The system will be implemented using Smalltalk, C++ and Java. The software will be supported on a UNIX machine, and orders will be stored in an Oracle database.

There will be an interface from the order taking facility to a warehouse management system supporting order fulfilment. There will also be an interface from the order taking facility to a product maintenance system and an interface to a bank to provide on-line credit authorisation.

#### 3.4.2 Use case model

The use case model contained 4 actors, 2 primary and 2 secondary, and 22 use cases. Both the actors and the use cases were described using a template as shown in Appendix A. The use cases were described with much detail, and they are similar to the use case models in the case study reported in [3].

#### 3.4.3 Development project

The system was developed over a period of 3 months, and 8 people were involved. The total effort was therefore approximately 24 person months. The development project was similar to the projects described in [3] in terms of length, total effort, number of people involved and also with regards to functionality.

## 4 Results

This section presents the estimates produced by the groups and the use case points method together with actual effort for the project.

### 4.1 Estimates

Table 4 shows which course the participants in the study attended and the number of people in each group. The table also shows what each group suggested as team size, elapsed time and total effort for the system to be constructed.

Some of the groups suggested a total effort in hours, days, weeks or years instead of months. The basis for the formula used to convert into person months was the following load: 7 hours pr. day, 5 days pr. week, 4 weeks pr. months and 11 months per year.

The author had exactly the same information about the development project as the groups, that is, the problem statement and the use case model, and spent approx. 10 minutes to produce an estimate using the use case points method. The value for each technical factor was assessed from the description in the problem statement. Detailed characteristics of the actual team that developed the system were unknown, the only information available was that there were no particular problems in terms of skills or motivation. Table 5 therefore has 2 use case points estimates. The first estimate, 29 person months, was produced by assigning the environmental factors F1-F6 the value 3, and the factors F7-F8 the value 2 under the assumption that the team was slightly better than average. The second estimate, 31 person months, was produced by omitting the environmental factors from the estimate.

Table 5 also shows actual effort for the project. The formula described in Section 4.1 was used to convert between person hours and person months. The use case points method does not suggest team size or elapsed time and the actual effort in person hours is not available. Therefore, the three corresponding cells are empty.

**Table 4. The estimates from each group**

| Group information | | | Suggestions | | |
|---|---|---|---|---|---|
| **Course no.** | **Group id.** | **Number of people** | **Team size** | **Elapsed time in months** | **Total effort in person months** |
| 1 | 11 | 4 | 5 | 6 | 20 |
| 1 | 12 | 4 | 5 | 8 | 30 |
| 1 | 13 | 3 | 4 | 4 | 16 |
| 2 | 21 | 4 | 9 | 4 | 33 |
| 2 | 22 | 4 | 3 | 4 | 11 |
| 2 | 23 | 3 | 5 | 3 | 15 |
| 2 | 24 | 3 | 3 | 5 | 12.5 |
| 3 | 31 | 3 | 6 | 2.25 | 11 |
| 3 | 32 | 3 | 8 | 5 | 35 |
| 3 | 33 | 3 | 5 | 5 | 22 |
| 3 | 34 | 3 | 5 | 4.2 | 12.5 |

**Table 5. Method estimate and actual effort**

| **Method** | **Team size** | **Elapsed time in months** | **Effort in person hours** | | **Effort in person months** | |
|---|---|---|---|---|---|---|
| Use case points | | | 4086 | 4370 | 29 | 31 |
| Actual effort | 8 | 3 | | | 24 | |

### 4.2 Discussion of the Results

The results in Tables 4 and 5 show that both the experts and the use case points method produced reasonably accurate estimates based on the available information. On average the groups of experts estimated a total effort of 19.8 person months, and the mean Magnitude of Relative Error (MMRE) is 0.37. The Magnitude of Relative Error for the use case points estimate is 0.21 or 0.29 depending on how the values for the environmental factors were assigned. However, we must expect inaccuracies in the estimates during the requirements phase. Boehm states that these inaccuracies range up to 60 percent or more [6].

The use case points method produced estimates that were closer to the actual effort than the estimates produced by 8 of the 11 groups of professional software developers. This shows that the estimation method successfully exploited the information in the use case model, that is, the abstraction level of the use case model must have been fairly appropriate. The groups of experts, on the other hand, did not know exactly how to use the information in the use case model. They chose different estimation strategies leading to estimates of various precision. The results therefore support the results in [3,12], which indicate that the use case points method may support expert knowledge in producing accurate estimates. In [3] the estimation method produced estimates that were almost as accurate as the estimates produced by experts. In the study reported in this paper, the estimation methods were more accurate than the groups. The results reported in [17] show that familiarity with the application domain and the technology is important to produce accurate estimates. An explanation why the method performed better in this study may thus be that the participants in this study on average were less experienced with the application domain and the technology than the estimators in [3].

## 5 Discussion of the Study

The author believes that the fact that the estimates were made by groups instead of individuals contributed to a realistic setting. In the actual organization, estimates are usually made in groups, and the discussion between the three or four members of each group forced them to explain their individual estimates and correct each other.

One may argue that 15 minutes, the time available for estimation, is short for producing a serious estimate. However, time constraints are common in studies of this kind, see for example the experiments reported in [15]. Note that in our study the participants spent quite a lot of time getting familiar with the requirements for the system to be constructed before the estimation was done.

A challenge when evaluating the precision of an estimation method compared with expert estimates is that it is seldom feasible to have several teams of software developers developing the same system. This was the case also in this study, the estimators were asked to estimate how long they themselves would take to complete the system, but their estimates were compared with the time it took another team.

The actual effort spent on the development project used in this study was 24 person months. The claims for the use case points method made in this paper are therefore based on the assumption that the estimation groups also would have implemented this system in approximately 24 person months. There are, however, some threats to this assumption. For example, several of the groups suggested a team size smaller than the 8 persons who were involved in the actual project. The team size may affect the effort of a project; typically a decrease in team size results in lower total effort. On the other hand, the participants in this study have less experience with this type of project and the technology used than the developers who implemented the project. This aspect makes it likely that they might have spent more time than the actual developers.

The estimate of a software development project often impacts the actual effort [8]. For example, a development team may want to fulfil an estimate if there is a lack of time even if this means implementing less functionality than originally specified. Therefore, if the actual effort for a project is very close to the estimated effort, this may not necessarily mean that the estimate was good [1].

The format and quality of the use case model may impact the estimates. In this study, only one use case model, described using one particular format, was used. We have, however, discussed how the format of the use case model impacts the estimates in a paper reporting three case studies on the use case points method [2]. There are various sets of guidelines on writing use cases, we have reported the results from an experiment evaluating how three different sets of guidelines affect the quality of the resulting use case model in [3].

## 6 Conclusions and Future Work

A study was conducted in a large international software development company with the aim of investigating how an estimation method based on use cases, the use case points method, performs compared with groups of experienced software developers.

The use case points method produced estimates that were quite close to the actual effort spent on a development project. They were closer to the actual effort than most of the estimates produced by the groups of professional software developers. The estimation method gave a best estimate with Magnitude of Relative Error (MRE) equal to 0.21, while the Mean Magnitude of Relative Error (MMRE) for the estimates made by the groups of experts was 0.37.

This supports earlier results indicating that when a use case model for a project is available, the use case points method may support expert knowledge. The results also show that the combination of expert estimates and method based estimates may be particularly beneficial when the estimators lack specific experience with the application domain and the technology to be used.

We are now continuing this work by investigating how the use case points method performs on different types of projects, in particular regarding size and level of detail in the use case model, and we are cooperating with several companies in order to evaluate the method on projects from different companies.

In parallel we are pursuing the work on how the use case points method best can be used in combination with expert knowledge. We are currently running a series of

interviews with project managers in a particular company. The aim of these interviews are twofold; to determine in what situations they feel the need for the support of an estimation method, and to find which elements of their current, informal estimation process can be formalized and used in combination with the use case points method.

A controlled experiment is planned with professional software developers to assess whether the use case points method should be applied also when a detailed use case model is not available. The aim of the experiment will be to examine to what extent and under what conditions professional software developers can correctly identify the number of use cases and their complexity from a textual requirements specification.

## Acknowledgements

Bruce Anderson and Paul Fertig made the course material for the course on use case modelling and made this research possible. Tor Svelle and Arne Kragh gave me the opportunity to teach the course. I thank Rune Markussen for teaching the course with me, and I thank the participants in the three courses. I also thank Magne Jørgensen, Dag Sjøberg and Ray Welland for useful comments on this paper.

## References

1. Abdel-Hamid, T.K. and Madnick, S.E. Lessons Learned from Modeling the Dynamics of Software Development. Communication of the ACM, 32(12), December, pp. 1426-1438, 1989.
2. Albrecht, A.J. Measuring Application Development Productivity. Proceedings of Joint SHARE, GUIDE, and IBM Application Development Symposium. 1979.
3. Anda, B., Dreiem, D., Sjøberg, D.I.K., and Jørgensen, M. Estimating Software Development Effort Based on Use Cases - Experiences from Industry. In M. Gogolla, C. Kobryn (Eds.): UML 2001 - The Unified Modeling Language.Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, LNCS 2185, Springer-Verlag, pp. 487-502.
4. Anda, B., Sjøberg, D.I.K. and Jørgensen, M. Quality and Understandability in Use Case Models. In J. Lindskov Knudsen (Ed.): ECOOP 2001 – Object – Oriented Programming. 15th European Conference, Budapest, Hungary, June 18-22, 2001, LNCS 2072, Springer-Verlag, pp. 402-428.
5. Blattberg, R.C. and Hoch, S.J. Database models and managerial intuition: 50% model + 50% manager, *Management Science,* Vol. 36, No. 8, pp. 887-899. 1990.
6. Boehm, B.W. *Software Engineering Economics*. Prentice-Hall. 1981.
7. Fetcke. T., Abran, A. & Nguyen, Tho-Hau. Mapping the OO-Jacobson Approach into Function Point Analysis. Technology of Object-Oriented Languages and Systems, TOOLS-23. IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 192-202. 1998.
8. Jørgensen, M. & Sjøberg, D. Impact of Software Effort Estimation on Software Work. Accepted for publication in Journal of Information and Software Technology. 2001.
9. Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21. December 1993.

10. Kitchenham, B. *Software metrics: measurement for software process improvement*. Blackwell Publishers. 1996.
11. Marchesi, M. OOA Metrics for the Unified Modeling Language. In Proc. of the Second Euromicro Conference on Software Maintenance and Reengineering, IEEE Comput. Soc, Los Alamitos, CA, USA; pp. 67-73. 1998.
12. Ribu, K. Estimating Object-Oriented Software Projects with Use Cases. Masters Thesis, University of Oslo. November 2001.
13. Schneider, G. & Winters, J. *Applying Use Cases – A Practical Guide*. Addison-Wesley. 1998.
14. Smith, J. The Estimation of Effort Based on Use Cases. Rational Software, White paper. 1999.
15. Stensrud, E. Empirical Studies in Software Engineering Management of Package-Enabled Reengineering Project. Ph.D. thesis, University of Oslo. 2000.
16. The Object Factory. Estimating Software Projects using ObjectMetrix, White paper. April 2000.
17. Vidger, M.R, & Kark, A.W. Software Cost Estimation and Control. National Research Council Canada, Institute for Information Technology. NRC No. 37116. 1994.
18. www.theobjectfactory.com

# Appendix A

| | |
|---|---|
| **Use Case id and name** | UC 001: Register new customer |
| **Scope & Level** | Primary use case for system for shopping through the internet |
| **Goal in context** | Register a new customer for access to the system for shopping through the internet. |
| **Preconditions** | None |
| **Primary actor** | Customer or call-centre operator |
| **Secondary actors** | None |
| **Main scenario** | 1. System prompts for customer post-code<br>2. Customer supplies post-code<br>3. System determines that customer is within the delivery area and prompts for customer details<br>4. Customer supplies name, postal address, telephone number and e-mail address.<br>5. System stores the details, issues a unique reference and confirms successful registration.<br>6. Use case ends successfully. |
| **Alternatives** | 3a. Customer post-code is outside delivery area<br>3a1. System informs customer that they are outside the eligible area.<br>3a2. Use case ends in failure.<br>5a. Customer already registered<br>5a1. System informs customer that name and address are already registered.<br>5a2. Use Case ends in failure.<br>5b. Customer blacklisted<br>5b1. System informs customer that registration has been rejected.<br>5b2. Use case ends in failure. |
| **Variations** | If e-mail address is not supplied by the customer, a generic call-centre e-mail address is inserted instead as a destination for order exception reports. |

*PAPER VI:*

# Improving Estimation Practices by Applying Use Case Models

*Bente Anda, Endre Angelvik and Kirsten Ribu*

M. Oivo and S. Komi-Sirviö (Eds): PROFES 2002 - 4th International Conference on Product Focused Software Process Improvement, Rovaniemi, Finland, December 9 - 11, 2002, pp. 383-397, LNCS 2559, Springer-Verlag.

## Abstract

An estimation method based on use cases, *the use case points method*, has given promising results. However, more knowledge is needed about the contexts in which the method can be applied and how it should be adapted to local environments to improve the estimation process. We applied the use case points method to several projects in a Scandinavian software development company as the first activity in a software process improvement project on improving estimation. The second activity of the improvement project was to conduct interviews with project managers and senior developers about how to obtain continued and more widespread use of the method in the company. Based on the interviews, we propose a tailored, potentially improved version of the method and suggest how estimation practices can be improved by applying it. We believe that these experiences may be of interest to other companies that consider applying use case models as part of their estimation practices

# 1 Introduction

A use case model describes the functional requirements of a system to be constructed, and use case models are frequently used as input to the process of estimating software development effort. An estimation method based on use cases, the use case points method, was introduced by Karner [9]. This estimation method has been evaluated in several software development projects with promising results [2,3,12]; it was considered easy to use and performed similar to or better than teams of very experienced software developers. Nevertheless, more knowledge is needed about how to apply the method and tailor it to a specific organization.

We evaluated the use case points method on three projects as the first activity in a software process improvement project on improving estimation in a Scandinavian software development company, Mogul [3]. Since then, the company has also applied the method on a couple of other projects with success.

The improvement project is conducted as part of a Norwegian software process improvement project, PROFIT, with the Universities of Oslo and Trondheim, SINTEF and 12 software development companies as partners. The goal of Mogul's improvement project is to develop an estimation method based on use case models that is simple to use and that is supplementary to expert estimates.

The second activity in the project was to conduct interviews with project managers and senior developers to

1. understand the ordinary estimation process in the company,
2. find out how the method for estimation based on use cases can be tailored to the company, and
3. establish the necessary context for applying the method successfully.

It is often difficult to sustain software process improvement projects beyond the initial phase, so the interviewees were also asked about how a supplementary method could obtain continued and widespread use in Mogul.

This paper describes Mogul's ordinary estimation process and its current practices for use case modeling. Then, contrasting the ordinary estimation process with evaluated best practices for estimation [8], areas of Mogul's estimation process are identified that may be improved by applying the use case points method. The paper also discusses requirements on the use case model that must be fulfilled for the use case points method to be applicable.

Context information is often missing when new or improved estimation methods are reported. The work described in this paper may provide a background for other companies that wish to improve their estimation practices applying use case models.

A major result of the interviews is a proposed modification of the use case points method, which includes, for example, an alternative way of measuring the size of a use case and modified adjustment factors.

The remainder of this paper is organized as follows. The use case points method is described in Section 2. Section 3 describes the context of the study. Section 4 describes estimation practices in Mogul and how they can be improved. Section 5

describes current practices for use case modeling. Section 6 suggests how the use case points method can be tailored to the company. Section 7 concludes and suggests future work.

## 2 The Use Case Points Method

The use case points method was initially developed by Gustav Karner [9]. It is based on the function points method [1], and the aim was to provide a simple estimation method adapted to object-oriented projects. This section gives the steps of the method as described in [13]. The method requires that it should be possible to count the number of transactions in each use case. A transaction is an event occurring between an actor and the system, the event being performed entirely or not at all.

1. The actors in the use case model are categorized as *simple*, *average* or *complex* depending on assumed complexity. A weight is assigned to each actor category:
   - Simple actor – another system with a defined API: weight = 1
   - Average actor – another system interacting through a protocol: weight = 2
   - Complex actor – a person interacting through a graphical user interface or web-page: weight = 3

   The total *unadjusted actor weight* (*UAW*) is calculated counting the number of actors in each category, multiplying each total by its specified weight, and then adding the products.

2. The use cases are correspondingly categorized as *simple*, *average* or *complex*, depending on the number of transactions, including the transactions in alternative flows. A weight factor is assigned to each use case category:
   - Simple use case – 3 or fewer transactions: weight = 5
   - Average use case – 4 to 7 transactions: weight 10
   - Complex use case – more than 7 transactions: weight 15

   The *unadjusted use case weights (UUCW)* is calculated counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The UAW is added to the UUCW to get the *unadjusted use case points* (*UUPC*).

3. The use case points are adjusted based on the values assigned to a number of technical factors (Table 1) and environmental factors (Table 2). These factors are meant to account for effort that is not related to the size of the task.

   Each factor is assigned a value between 0 and 5 depending on its assumed influence on the project. A rating of 0 means that the factor is irrelevant for this project; 5 means that it is essential.

*The technical complexity factor (TCF)* is calculated multiplying the value of each factor in Table 1 by its weight and then adding all these numbers to get the sum called the *TFactor*. Finally, the following formula is applied:

$$TCF = 0.6 + (.01*TFactor)$$

*The environmental factor (EF)* is calculated accordingly by multiplying the value of each factor in Table 2 by its weight and adding all the products to get the sum called the *Efactor*. The following formula is applied:

$$EF = 1.4 + (-0.03*EFactor)$$

The *adjusted use case points (UCP*) are calculated as follows:

$$UCP = UUCP*TCF*EF$$

4. The number of person hours per use case point for a project estimate is determined by the environmental factors because these are considered to have a large impact on the actual effort [13]. The number of factors in F1 through F6 that are below 3 are counted and added to the number of factors in F7 through F8 that are above 3. If the total is 2 or less, 20 person hours per UCP is used; if the total is 3 or 4, 28 person hours per UCP is used. If the number exceeds 4, it is recommended that changes should be made to the project so the number can be adjusted, or alternatively that the number of person hours should be increased to 36 per use case point.

A spreadsheet is used to implement the method and produce an estimate. The method provides an estimate in total number of person hours.

The use case points method can be criticized from a theoretical point of view as has the function points method. The addition and subsequent multiplication of ordinal values, for example, is theoretically invalid [10]. However, the function points method has shown to predict effort reasonably well for many types of systems.

There are several other methods for use case based estimation. The methods differ in that size and complexity of the use cases are measured differently, and in that different technical and environmental factors are considered. Two alternative methods for estimation based on use cases are described in [6,14]. The method described in [6] maps attributes of the use case model into function points. In [14] a certain number of lines of code is assumed for each use case, and the total number of lines of code is used as a basis for the estimate. *Tassc:Estimator* is a commercial tool for estimation based on use cases [17]. A metric suite for use case models, which can be used for estimation, is suggested in [11], but a complete estimation method is not presented.

**Table 1. Technical complexity factors**

| Factor | Description | Wght |
|---|---|---|
| T1 | Distributed system | 2 |
| T2 | Response or throughput performance objectives | 2 |
| T3 | End-user efficiency | 1 |
| T4 | Complex internal processing | 1 |
| T5 | Reusable code | 1 |
| T6 | Easy to install | 0.5 |
| T7 | Easy to use | 0.5 |
| T8 | Portable | 2 |
| T9 | Easy to change | 1 |
| T10 | Concurrent | 1 |
| T11 | Includes Security features | 1 |
| T12 | Provides access for third parties | 1 |
| T13 | Special user training facilities are required | 1 |

**Table 2. Environmental factors**

| Factor | Description | Wght |
|---|---|---|
| F1 | Familiar with Rational Unified Process | 1.5 |
| F2 | Application experience | 0.5 |
| F3 | Object-oriented experience | 1 |
| F4 | Lead analyst capability | 0.5 |
| F5 | Motivation | 1 |
| F6 | Stable requirements | 2 |
| F7 | Part-time workers | -1 |
| F8 | Difficult programming language | -1 |

# 3 Context of Study

This section gives some characteristics of the company we studied, presents the results from the former case studies conducted to evaluate the use case points method, and describes the interviews with senior personnel of the company.

## 3.1 The Company

Mogul is a medium sized Scandinavian software development company located in Norway, Sweden and Finland. In Norway there are approximately 180 employees. The business area is software development for public and private sector, in particular banking and finance. Mogul's projects can roughly be divided into two types: traditional software development projects based on a three-layer architecture and web- projects, that is, intranet, internet or extranet solutions. The web-projects often consist in adapting existing systems to a web-environment. The company takes responsibility for complete projects or sell hours as consultants or mentors on methods and architecture. Mogul gives courses on the Rational Unified Process (RUP), which is also used in their own projects whenever possible.

### 3.2 Results from Case Studies

The use case points method has been evaluated in 3 development projects in Mogul. The estimates produced with the use case points method were compared with expert estimates and actual effort. The results were promising in that the estimates provided by the method were as accurate as the average estimates of the projects in the company. Table 3 shows some characteristics of the case studies. Table 4 gives the results.

### 3.3 The Interviews

The interviewees, 1 administrative manager, 7 project managers and 3 senior developers, had from 6 to 26 years experience with software development, and were chosen because they were very experienced estimators.

The interviews were semi-structured with open-ended questions to allow the respondents to speak more freely of issues they felt were important. They were conducted by one or two interviewers, lasted from 45 – 60 minutes and were tape recorded.

**Table 3. Characteristics of three software development projects**

| Characteristic | Project A | Project B | Project C |
|---|---|---|---|
| Size | 7 months elapsed time, 4000 staff hours | 3 months elapsed time, 3000 staff hours | 4 months elapsed time, 3000 staff hours |
| Software architecture | Three-tier, established before the project | Three-tier, known, but not established in advance | As project B |
| Programming environment | Java (Visual Café and JBuilder), Web Logic | MS Visual Studio | Java (Jbuilder), Web Logic |
| Project members | 6 developers with 0 to 17 years experience | 6 developers with 0 to 12 years experience | 5 developers with 2 to 10 years experience, 4 consultants were involved part time. |
| Application domain | Finance | CRM (Customer relationship management within banking), part of a larger solution | Banking (support for sale of credit cards) |

**Table 4. Expert estimate, use case estimate and effort (in hours)**

| Project | Expert estimate | Use case estimate | | Actual effort |
|---|---|---|---|---|
| A | 2730 | 2550 | | 3670 |
| B | 2340 | 3320 | 2730[10] | 2860 |
| C | 2100 | 2080 | | 2740 |

[10] The first estimate for project B, 3320 hours, was produced based on information about actors and use cases given by the project manager. In the second estimate, 2730 hours, several actors with a very similar user interface were generalized into one super actor, and included and extending use cases were omitted.

# 4 Estimation Practices and Possible Improvements

This section describes current practices for estimation in Mogul based on the information from the interviews. The estimation practices are compared with best practices for estimation described in the literature to identify particular areas that may benefit from applying use case based estimation.

The two types of projects in the company are estimated differently, and are therefore treated separately below.

## 4.1 Estimating Traditional Software Development Projects

A project manager is responsible for producing a first estimate early in the inception phase. He/she may gather a team for the estimation process, but the actual developers are usually not allocated at this stage. The estimate indicates the need for resources, often together with a completion date. RUP gives generally good opportunities for negotiating with the client about functionality; specified functionality is frequently changed and given new priorities along the way. It is also often possible to get more resources if necessary. The completion date, however, is often critical.

The estimate is typically based on a requirements specification from the client, possibly with a solution outline and some use cases. Several of the interviewees also develop a high-level use case model, based on the available information, which in turn is also used in the estimation process.

Some estimates are made in offer situations where Mogul is bidding to get a project. In such situations only the client's description of the functionality is available; and it is difficult to get more information. The company therefore depends on the clients' ability to describe what they actually want.

If the project mainly involves new development, Mogul's policy is to conduct a pre-project to clarify the requirements and construct a detailed use case model before committing to an estimate. However, the client often wants to know what kind of solution can be had for the price they can afford without paying for a pre-project, and it may therefore be difficult to avoid giving an early estimate based on insufficient information. One of the interviewees describes this situation using the analogy of buying a car: "You have all sorts of requirements for your new car, but you only have € 5000, so you wish to know what you can get for that amount of money".

The estimation process is bottom-up because the project is broken down into activities that are estimated separately, perhaps by different people. Sometimes two people are involved in estimating the same activity, either discussing to reach an estimate, or by letting an independent person go through the estimate afterwards. Mostly, however, estimation is done individually, and estimates for different parts are added to form the complete estimate. Several of the interviewees had their own methods or spreadsheets to help them in the estimation process.

The ability to identify risks is an important part of estimation. The interviewees claimed to be good at identifying technological risks, but believed themselves to be less good at identifying organizational risk.

The time for project management, in the order of 5-15%, is added to the estimate. The estimate must also take into account that much of the developers' time is spent on

other activities such as meetings. The percentage of the developers' time believed to be available for development varied among the interviewees from 50% to 80%.

It may also be sensible to consider whether the client is in public or private sector. This may impact effort because more people tend to be involved in the decision process in the public sector. Expected lifetime for the system should also be considered because this has implications for the documentation and subsequently for the effort.

New estimates are usually produced in the elaboration phase, typically after the first iteration. The developers re-estimate their bits, for example, screens or modules and assess how much time is needed for completion.

Mogul does not keep track of the accuracy of its estimates, so it is impossible to assess the typical precision of their estimates. The interviewees stated, however, that the estimates are usually overrun.

### 4.2 Estimating Web-projects

The web-projects differ from the traditional development projects in that they are smaller, they more often build on an existing solution, and the functionality is less complicated. The most important part of these projects is establishing the information structure. According to the interviewees, 40% of the resources are typically used on this activity. An outline of a graphical design is a prerequisite for an estimate. The effort put into the graphical design will vary based on how much the client is willing to pay. A solution will also include a number of templates into which the users will fill in information. Each template typically takes one day to develop.

At present, estimating these projects is not difficult, but some of the interviewees expected the two types of projects to merge as traditional software projects start include advanced web interfaces.

### 4.3 Improving Estimation Practices

We have compared the ordinary estimation practices in Mogul with best practice principles for estimation [8] to identify how the use case points method can improve the estimation practices and thereby the accuracy of the estimates. Below we describe the best practice principles that are relevant in our context and how they can be fulfilled:

1. "Ask the estimators to justify and criticize their estimates."
   A supplementary use case based estimate may, if it differs from the expert estimate, provide a basis for criticizing the expert estimate.
2. "Estimate top-down and bottom-up, independently of each other."
   The company's expert estimates are made bottom-up. The use case points method, on the other hand, provides a top-down estimate. A top-down estimate is produced identifying some characteristics of the development project and using those as input to a complete estimate.
3. "Combine estimates from different experts and estimation strategies."

It has been shown sensible to combine models and human judgment [5], but more work is needed on how to best combine expert estimates and estimates produced with the use case points method.

4. "Assess the uncertainty of the estimate."
   The spreadsheet used to produce an estimate with the use case points method makes it possible to vary the input both with regards to the number and size of the use cases and with regards to the different technical and environmental factors. This may help assess uncertainty due to unknown factors in the development project.

The use of an estimation method in combination with expert estimates can also lead to the avoidance of biases and large overruns, and estimation methods have been shown to perform better than expert estimators with little domain experience [7,8]. Therefore, the support given by an estimation method may make more people competent to take part in estimation.

# 5 Practices for Use Case Modeling

To be suitable as a basis for estimation, a use case model should be correct and described at an appropriate level of detail. This section gives a brief overview of how use case modeling is done in Mogul, and discusses challenges relating to correctness and level of detail of the use cases.

In Mogul, use case modeling is applied in traditional software development projects to identify and describe business logic. Use case modeling is usually not applied in web-projects because use cases lack the possibility to describe functionality where a web interface lets the user perform a function by switching among different web pages or where it is necessary to save current work and later resume it. Another problem is that the terminology in RUP is unfamiliar to several of the participants that typically take part in web-projects, for example, graphical designers.

Moreover, the use cases are perceived as belonging to and driving development projects; they are seldom maintained in the elaboration phase and never when the system has become operational. Therefore, the original use cases are often outdated and unsuitable as a basis for specifying modified functionality in maintenance projects.

## 5.1 Use Case Modeling Process

The use case modeling process in Mogul is as follows. In the inception phase, use case models may just be described at a high level without details. It may supplement the client's requirements specification or be derived from it.

A detailed use case model is usually constructed as part of a pre-project together with representatives of the client. The use case modeling process is a breadth-first process where the first activity is to identify actors and use cases, and then construct a use case diagram. Subsequently, the use cases are detailed out, possibly in several iterations. The participants from Mogul set up the structure, while the participants

from the client fill in the details. The participants work individually on the different use cases and meet regularly to discuss them. The use cases may also be constructed solely by the clients. The use cases are often supplemented by screens and a domain model.

Pen and paper are often used to construct the use cases, and then Rational Rose is used to document the use case diagram and different templates, depending on the project, are used to document the use case descriptions. Some of the interviewees also use the add-on tools to Rational Rose, Requisite Pro or SODA, to document the use cases.

When the use case model is completed, the project participants, in particular those from the client, often read through the use case model to verify that the requirements are covered.

### 5.2 Correctness of the Use Case Model

A use case model should be correct in that the functional requirements of all user groups are included. The interviewees found the use case modeling useful because it helps focus on functionality from the point of view of the user and helps assure that the requirements of all the user groups are included. They also found the technique useful for obtaining a common understanding of the requirements and for reaching agreement with the client.

The use case modeling process can often be a maturity process for the clients; they are forced to think through what they actually want. One of the interviewees described it like this: "The clients' domain expert thought she had a good overview of the requirements, but because of the use case modeling process we found out that not everybody agreed with her about what should be the functionality of the system."

It may, however, be difficult to find end-users with sufficient competence and interest to participate in use case modeling. Some of the interviewees meant that use cases were too abstract for end-users. End-users may also be confused by the sequential description of the steps of the individual use cases and believe that the sequence must be followed strictly. They may also find it difficult to understand from the use case model how the individual use cases relate.

### 5.3 Level of Detail of the Use Cases

A balanced level of detail in the use cases is important when the use case model is to be used as a basis for estimation. If the use cases are unbalanced, there may be difficulties when measuring the size of the use cases with the use case points method.

The interviewees found it difficult to balance the use cases. In their opinion, use case descriptions tend to include too much detail. One of the interviewees described the problem in the following way: "The use cases tend to expand infinitely because to get complete descriptions we keep discussing unimportant details for a long time." The proposed solution to this problem is to have good examples of use case models available, and to use tabular descriptions of the use cases to avoid too much text.

Another solution may be to use specific guidelines in the use case modeling process as proposed in [4].

Since use cases describe functionality from the point of view of the end-users, they seldom provide sufficient architectural information, and the descriptions may hide complex business logic. These issues are described further in the next section.

# 6 Adapting the Use Case Points Method

The interviewees had experience from estimation based on use cases, and had suggestions for tailoring the use case points method, both with regards to measuring size (Section 6.1) and with regards to which technical and environmental factors were relevant in this particular company (Section 6.2). Section 6.3 discusses how to estimate architecture when the use case points method is applied. Section 6.4 suggests how the use case points method can be more widespread in Mogul.

## 6.1 Assessing Size of the Use Cases

The use case points method takes the size of each use case as input. Size is measured in number of transactions in the use case descriptions. According to the interviewees, there are some problems with this measure:

- It is desirable to estimate with the use case points method in the inception phase, but at this stage the use cases may not sufficiently detailed out to show the individual transactions.
- When the use case descriptions are detailed out, they may be described at an unbalanced level of detail, which in turn may lead to skewed results due to inaccurate measure of size.
- The size measure does not capture complexity in the business logic and the architecture that may be hidden in the use case descriptions.

As a response to these difficulties, the interviewees suggested alternative ways of measuring size, for example, that weights could be assigned to each use case based on the intuition of the estimator or that the use cases could be used as a basis for identifying components to be estimated. However, these suggestions may contradict our goal of developing a method that requires little expert knowledge.

The following method was suggested by one of the interviewees as a supplement to counting transactions.

Consider for each use case what has to be done in the presentation layer, the persistence layer and the business layer:

1. The effort in the presentation layer will depend on the number of new screens, the number of transfers from one screen to another, the number of forms in the screens and the number of places where dynamic content must be generated.
2. The effort in the persistence layer will depend on the impact on the data model and persistent data, that is, on the number of new tables, the number of changes to table definitions, and the number of queries and updates on the tables.

3. The effort in the business layer is difficult to quantify as it may be anything from input to a database to complicated data processing, possibly also access to different back-systems. One of the interviewees described it this way: "The business logic may just be about transferring data, but you may find that you need a lorry to actually do it". Our advice is, therefore, that the estimators should break down each use case sufficiently to form an opinion about the complexity of the business logic necessary for realizing it. If this is impossible, alternative estimates could be made for the most likely and the most pessimistic size of the use cases.

### 6.2 Adjustments Factors

In the use case points method, the estimate based on the size of the use cases is adjusted based on a number of technical and environmental factors. The method is inspired by the function points method, particularly the MkII function point analysis (MKII FPA) [15]. The two methods use several of the same technical factors. The technical factors of MkII FPA, however, have since been discarded [16]. They may be relevant early in a project's life-cycle when the requirements are stated in a general form, but when the requirements are detailed out, many of them will have influenced the functional requirements, so that adjusting the effort using the technical factors may lead to double counting. In [10] evidence is also presented that the adjustment factors applied in the function point method are unnecessary, particularly when the method is used in a single organization. In a case study, the use case points estimates for five projects were on average more accurate when the technical factors where omitted [12]. We therefore propose that the technical factors be omitted when the use case points method is applied to detailed use cases.

The environmental factors are not taken into account by the detailed use case descriptions and should therefore be considered. Some environmental factors may, however, be irrelevant to this particular company, and it may be necessary to consider other factors. The environmental factors regarding the development team, F1 – F6, were all considered relevant by the interviewees. Nevertheless, they stated that it would be beneficial to specify productivity and availability for each team member, instead of having to calculate an average, because there are large differences in productivity among developers. The interviewees also felt that they were usually too optimistic about the productivity of the team members. Regarding availability, many of the company's projects are located at the clients, which means that they are "at the mercy of the clients" regarding their ability to provide people with necessary knowledge about the application domain and technological infrastructure. The environmental factors may also be useful to show the client the consequences of uncertainties and risks in the project.

Requirements stability, F7, was considered irrelevant when using RUP, because one of the primary motivations for using RUP is that it gives the possibility to continually change the requirements.

Difficulty of the programming language, F8, was considered difficult to assess and therefore irrelevant because the development projects now require that the developers have knowledge about the technology used at each layer in the architecture.

## 6.3 Functionality versus Architecture

The interviewees meant that architecture mostly should be estimated separately from functionality: "The whole project can be estimated based on use cases only if you know the customer and the architecture well from previous projects, but if there is much uncertainty, the architecture should definitely be estimated separately."

Our goal is to develop a method that can provide a complete estimate, which requires that it can estimate a new or modified architecture. We therefore propose that if an architecture already exists, the impact on the architecture should be considered for each use case and be used to adjust the size measures based on number of transactions.

We also propose, as did one of the interviewees, that the environmental factor F7, could be used to assess the architecture. A value of 5 (meaning new architecture or major changes to existing architecture) assigned to F7 increases the estimate by approximately 60% compared with the estimate produced when the value of F7 is 0 (meaning existing and stable architecture). One problem with this solution is, however, that the percentage of effort added for architecture is the same independently of the size of the project. In the interviewees' opinion, the proportion of the effort required for the architecture compared with the effort required for the functionality varies with the size of the project; the larger the project, the smaller is the proportion of effort needed to establish the architecture. One of the interviewees explained that many of the activities to establish the architecture must be done whether there are 5 or 50 use cases. He also mentioned as an example a project that took 8 months, and where 1/3 of the effort was on architecture and 2/3 on functionality. In a smaller project that took 3 months, 1/2 of the effort was spent on architecture and 1/2 on functionality.

## 6.4 Widespread Use of the Use Case Points Method in Mogul

The use case points method has been applied to several projects in Mogul. Nevertheless, obtaining continued and more widespread use of the method remains a challenge. We therefore wanted the interviewees' opinion about the prerequisites for a successful use of the use case points method in a larger scale. Our interviewees tended to use various tools for use case modeling, and they also used various tools and spreadsheets in estimation. This may indicate that there is a culture for applying tools and methods in an ad-hoc way in the company. Some of the interviewees stressed that they wanted a tool to be applied when they themselves found it useful, not methods that they were forced to apply. Hence, it may be difficult to get the whole company to agree on applying the use case points method.

Nevertheless, the interviewees were positive towards applying the use case points method; they found it desirable to apply the use case models in more activities in the development projects because of the effort that is often put into making it. A method to supplement expert estimates was considered particularly useful in projects with much uncertainty.

Although we agree that the use of the use case points method should be voluntary in Mogul, more experience with the method is needed to make it generally applicable.

## 7 Conclusions and Future Work

As part of a former software process improvement work in the software development company Mogul, an estimation method based on use cases, the use case points method, was evaluated with promising results. This paper described a follow-up software process improvement work that included interviews with senior personnel of Mogul to establish how the use case points method could improve the company's estimation practices, the prerequisites for applying the method and how to tailor it to this particular company.

We found that the use case points method can improve estimation practices in Mogul in that it provides a supplementary estimate in addition to the expert estimate. Combining estimates from different estimation strategies, particularly combining bottom-up estimates with top-down estimates, is an evaluated principle for improving estimates. In addition, applying an estimation method may help avoid estimation biases and thereby large overruns.

We also found that even though Mogul has good knowledge of RUP and use case modeling, it is challenging to construct a use case model that forms a good basis for estimation in that it correctly describes the functionality of the system and that the use cases are balanced. In particular, it is difficult to find end-users with sufficient competence and interest to take part in use case modeling. Nevertheless, the interviewees found use case models superior to old, unstructured requirements specifications.

The use case points method requires that the use cases should be detailed out, that is, each event between the system and the actor should be described, but this is not always done. We therefore proposed how the assessment of size of each use case could be refined, and made some suggestions for how the technical and environmental factors in the use case points method can be applied successfully to estimate the company's projects.

Nevertheless, more work is needed on how to tailor the use case points method. The following activities are planned:

- Establishing a scheme for measuring improvement to the estimation process. The most obvious success criterion is the accuracy of the estimates. Another criterion may be the number of people in the company who are competent estimators.
- Conducting a follow-up study to evaluate the proposed modifications to the use case points method.
- Investigating further how estimates produced with the use case points method can be combined with expert estimates.
- Investigating how use case modeling can be applied in web-projects.
- Investigating how to measure the size of a change to a use case, enabling the use case points method to be used in maintenance projects.

## Acknowledgements

We gratefully acknowledge the employees of Mogul who took part in the interviews. We also acknowledge Dag Sjøberg for valuable comments on this paper. This research is funded by The Research Council of Norway through the industry-project PROFIT (PROcess improvement For the IT industry).

## References

1. Albrecht, A.J. Measuring Application Development Productivity. Proceedings of joint SHARE, GUIDE and IBM Application Development Symposium. 1979.
2. Anda, B. Comparing Use Case based Estimates with Expert Estimates. Proceedings of the 2002 Conference on Empirical Assessment in Software Engineering (EASE 2002), Keele, United Kingdom, April 8-10, 2002.
3. Anda, B., Dreiem, H., Sjøberg, D.I.K., and Jørgensen, M. Estimating Software Development Effort Based on Use Cases – Experiences from Industry. UML'2001 - 4th Int. Conference on the Unified Modeling Language, Concepts, and Tools, Toronto, Canada, October 1-5, 2001, LNCS 2185, Springer-Verlag, pp. 487-502.
4. Anda, B., Sjøberg, D.I.K. and Jørgensen, M. Quality and Understandability in Use Case Models. ECOOP'2001, June 18-22, 2001, LNCS 2072 Springer-Verlag, pp. 402-428.
5. Blattberg, R.C. and Hoch, S.J. Database models and managerial intuition: 50% model + 50% manager, *Management Science*, Vol. 36, No. 8, pp. 887-899. 1990.
6. Fetcke, T., Abran, A. & Nguyen, T-H. Mapping the OO-Jacobson Approach into Function Point Analysis. Technology of Object-Oriented Languages and Systems, TOOLS-23. IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 192-202. 1998.
7. Jørgensen, M. An empirical evaluation of the MK II FPA estimation model, Norwegian Informatics Conference, Voss, Norway. 1997.
8. Jørgensen, M. Reviews of Studies on Expert Estimation of Software Development Effort. Submitted to Journal of Systems and Software.
9. Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21. December 1993.
10. Kitchenham, B. A. *Software Metrics: Measurement for Software Process Improvement*. Blackwell Publishers. 1996.
11. Marchesi, M. OOA Metrics for the Unified Modeling Language. In Proc. of the Second Euromicro Conference on Software Maintenance and Reengineering, IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 67-73. 1998.
12. Ribu, K. Estimating Object-Oriented Software Projects with Use Cases. Masters' Thesis, University of Oslo. November 2001.
13. Schneider, G. & Winters, J. *Applying Use Cases – A Practical Guide*. Addison-Wesley. 1998.
14. Smith, J. The Estimation of Effort Based on Use Cases. Rational Software, White paper. 1999.
15. Symons C.R. *Software Sizing and Estimating, MKII FPA*. John Wiley and Sons, 1991.
16. Symons, C. Come back function point analysis (modernized) – all is forgiven! Software Measurement Services Ltd. 2001.
17. http://www.tassc-solutions.com/

*PAPER VII:*

# Applying Use Cases to Design versus Validate Class Diagrams – A Controlled Experiment Using a Professional Modelling Tool

*Bente Anda and Dag I.K. Sjøberg*

Simula Research Laboratory Technical Report no. 2003 - 01, 2003.

## Abstract

Several processes have been proposed for the transition from functional requirements to an object-oriented design, but these processes have been subject to little empirical validation. A use case driven development process, where a use case model is the principal basis for a design model, is often recommended when applying UML. Nevertheless, it has been reported that this process leads to problems, such as the developers missing some requirements and mistaking requirements for design. This report describes a controlled experiment, with 53 students as subjects, conducted to investigate two alternative processes for applying a use case model in an object-oriented design process resulting in a class diagram. One process was use case driven, while the other was a responsibility-driven process in which the use case model was applied as a means of validating the resulting class diagram. Half of the subjects used the modelling tool Tau UML Suite from Telelogic; the other half used pen and paper. The results show that the validation process led to class diagrams implementing more of the requirements. The use case driven process did, however, result in class diagrams with a better structure. The results also show that those who used the modelling tool spent more time on constructing class diagrams than did those who used pen and paper. We experienced that it requires much more effort to organize an experiment with a professional modelling tool than with only pen and paper.

# 1 Introduction

There are several approaches to making the transition from functional requirements to an object-oriented design; examples are grammatical analysis of the requirements, common class pattern, Class Responsibility Collaboration (CRC) cards and the use case driven process [17]. In addition, an inspection technique that applies a use case model to validate design models has been proposed and empirically validated [21].

The functional requirements of a software system can be captured and documented in use cases, and a use case driven process in which the use case model is a primary artefact in the identification of system classes, is frequently recommended together with the UML [2,4,5,11,12,14,15,19]. This is claimed to ensure traceability between the different models describing the system. However, use case driven development processes have been criticized for not providing a sufficient basis for the construction of class diagrams, which is commonly used to represent object-oriented designs. For example, it is claimed that such a development process leads to:

- a gap between the use case model and the class diagram [19],
- missing classes because the use case model is insufficient for deriving all necessary classes, and
- the developers mistaking requirements for design as a use case description may show only one of several ways of achieving the goal of the use case [24].

An alternative to a use case driven process is to apply a use case model to validate an object-oriented design that is constructed by one of the other approaches. In the following, the term *validation process* is used to denote such development processes.

The different approaches have, to our knowledge, been subject to little empirical validation. In our opinion there is a need for more knowledge about which approach is the most appropriate in a specific context, for example, in terms of characteristics of the team that will use the process and the system to be constructed. Our goal is therefore to investigate empirically the advantages and disadvantages of different ways of applying a use case model in an object-oriented design process.

We previously conducted a pilot experiment to compare a use case driven process with a validation process [25]. Based on the results, we designed the controlled experiment reported here formulating hypotheses to investigate whether the two processes result in differences in the quality of the resulting design with regards to completeness and structure. We also investigated whether the two processes are different with respect to the time required for constructing a class diagram. The hypotheses were tested with 53 students as subjects. The task was to construct a class diagram for a library system.

In the pilot experiment, the subjects only used pen and paper. To increase the realism of the context [10,22] in this experiment, half of the subjects used the professional modelling tool Tau UML Suite from Telelogic [26], and the other half used pen and paper. The authors have found no other controlled experiment in the field of object-oriented design in which the subjects used a professional modelling

tool to support the design process. Hence, one of the purposes of this study was to gain experience of conducting controlled experiments with such a tool.

The results show that the validation process led to more complete class diagrams, that is, they implemented more of the requirements. The use case driven process did, however, result in the class diagrams being structured better.

The results further show that those who used the modelling tool spent more time on constructing class diagrams than did those who used pen and paper. Our experience is also that it requires more effort to organize an experiment that involves the use of a modelling tool.

The remainder of this paper is organized as follows. Section 2 gives an overview of different processes for the transition from requirements to object-oriented design and describes the two processes evaluated in this experiment in detail. Section 3 presents the experimental design. Section 4 presents the results. Section 5 discusses the use of a modelling tool in the experiment. Section 6 presents some threats to the validity of the results. Section 7 concludes and suggests further work.

## 2 Transition from Use Case Model to an Object-Oriented Design

In the UML meta-model, a use case is a subclass of Classifier [27]. A use case specifies the sequences of actions that the use case should be able to perform, that is, changes of state and communications with the environment. This implies that a use case contains a state and behaviour from which classes, attributes and methods can be derived. The use case package, which consists of actors and use cases, and which, therefore, can be considered as equivalent to a use case model, is a sub-package of the Behavioral Elements package in the meta-model. This means that a use case package defines the behaviour of a system without revealing its internal structure.

The first step of the transition from a use case model to an object-oriented design is typically to identify a set of analysis classes. The analysis class diagram is then elaborated upon to produce a design model from which code can be generated.

There are several approaches to identifying classes from use cases and other requirements specifications [17]:

1. In grammatical analysis the requirements specification is searched for *nouns*, which are candidate classes or attributes to classes, and *services* to be delivered by the system, which are candidates for methods. If the emphasis is on searching the requirements specification for nouns, the approach can be characterized as data-driven [20]; if the emphasis is on services, however, the approach can be classified as responsibility-driven [28].
2. Common class pattern is based on the identification of the different kinds of classes of which a system will typically consist. Examples are physical classes, business classes, logical classes, application classes, computer classes and behavioural classes.
3. CRC cards are specially prepared for use in brainstorming sessions. Each developer plays one or more cards. New classes are identified from the message passing between the players. This is a responsibility-driven approach [28], which requires developers/participants who know the system requirements well.

4. The use case driven approach is frequently recommended together with the UML [2,4,5,11,12,14,15,19]. Sequence and/or collaboration diagrams are made for each use case scenario, and the objects used in these diagrams, as well as those of a domain model, lead to the discovery of classes. A variant of the use case driven approach, in which the classes are identified from the goals of each use case instead of from the scenarios, is suggested in [16].

We have compared the use case driven process with an alternative process that we propose, called a validation process. It is based on a process that was suggested as an alternative to a use case driven process: a responsibility-driven grammatical analysis used to identify the services to be implemented [18]. We chose a responsibility-driven approach instead of a data-driven one, because an evaluation of those two approaches showed that the responsibility-driven approach produced less complex designs than those produced by the data-driven approach [20]. The validation of the class diagram is based on the reading techniques for inspecting quality and consistency of diagrams presented in [21]. These processes were chosen because we wanted to focus on processes that apply use case models.

Another example of the application of use cases, and their corresponding sequence diagrams in the validation of class diagrams, is presented in [8]. Using that approach, changes in the use case model are detected automatically and are used as a basis for deriving test cases. The test cases are subsequently used to validate the class diagrams.

Figure 1 shows the steps of a use case driven process, while Figure 2 shows a responsibility driven process in which a use case model is applied to validate the design. The main difference between the two processes is in the identification of methods. In the use case driven process, sequence diagrams are used to identify the message passing between the classes; whereas in the validation process, methods are identified from grammatical analysis of the requirements specification, and the method composition is subsequently validated using sequence diagrams. The two processes and our motivation for evaluating them are discussed in more detail in [25].

In our experience, few organizations apply a use case model in a completely systematic way in their development process; frequently, the elements of several approaches are involved. The choice of development process is, of course, guided by a number of factors, for example, the analysts' overall knowledge and experience, the problem domain and existing architecture. Therefore, the processes described and evaluated in this paper represent recommended practice more than actual practice, but we believe that recommended practice should be subject to evaluation before becoming actual practice. The evaluation of a recommended software development process will give indications of its strengths and weaknesses, and about when the process is particularly suitable, which may facilitate its transfer into actual practice.

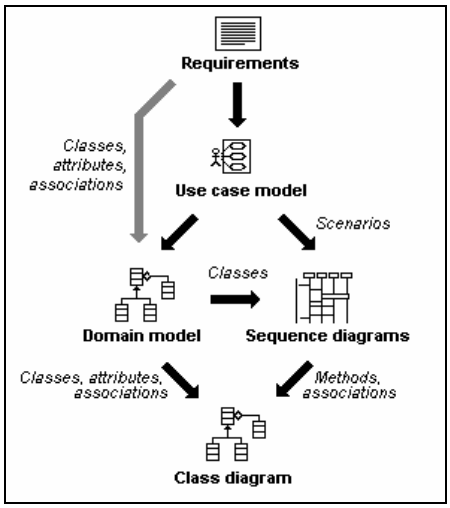


**Fig. 1. The use case driven process**

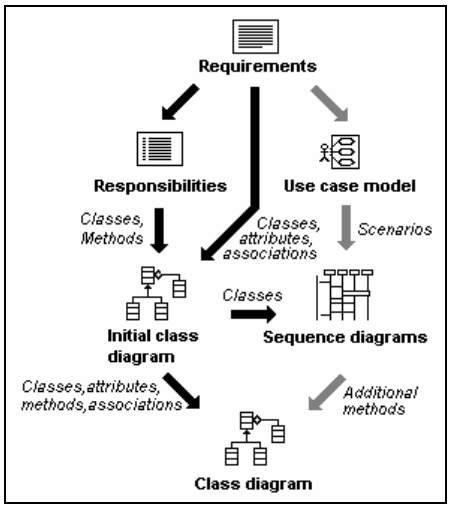


**Fig. 2. The validation process**

## 3 Design of Experiment

This section describes the experimental design, that is, the hypotheses tested and the evaluation scheme, as well as the subjects, the experimental material and the procedure of the experiment.

### 3.1 Hypotheses

The results from the pilot experiment [25] showed a difference both in the quality of the resulting class diagrams and in the time spent on design. To compare the two development processes, we tested the following hypotheses:

**H1**$_0$: There is no difference in the completeness of the class diagrams.
**H2**$_0$: There is no difference in the structure of the class diagrams.
**H3**$_0$: There is no difference in the time spent constructing the class diagrams.

### 3.2 Evaluation Scheme

The two design processes can be evaluated in terms of quality attributes of the resulting class diagrams, and in terms of direct quality attributes of the processes

themselves. This section describes how the resulting class diagrams and the design processes were evaluated.

The class diagram should capture all data and functional requirements, and also satisfy criteria for object-oriented design [3]. Therefore, the quality of the class diagrams was evaluated according to two dimensions:

1. *Completeness*, measured in terms of how much of the functionality described in the requirements specification was actually implemented in the class diagram. The following aspects should be satisfied:
   - All services described in the requirements specification are implemented.
   - The services are allocated to all and only the correct classes, that is, the class diagram contains all required correct classes and no superfluous classes.
   - The classes contain the necessary information in terms of attributes.

   Each class diagram was given a score between 0 and 5 on completeness.
2. *Structure*, measured in terms of high cohesion and low coupling. Cohesion and coupling were measured subjectively because the class diagrams were too small to apply established metrics, such as the high-level design metrics described in [6,7]. Each class diagram was given a score between 0 and 5 on structure.

The two direct process quality attributes evaluated were:

1. *Time* spent on creating the class diagrams.
2. *Process conformance*, measured in the number of subjects who managed to follow each of the two processes.

### 3.3 Subjects

The subjects were 53 students in an undergraduate course in software engineering. The students were in their 3rd or 4th year of study. They had learned the basics of object-oriented programming and UML through this and previous courses in computer science. They had also used the Tau UML Suite in this and one previous course. The experiment was voluntary, but they were informed that it was relevant for their course. They were paid for their participation.

The subjects can be considered a convenience sample, but we believe that they are representative of other students and perhaps also of junior professionals with limited experience with UML.

Prior to the experiment, the subjects were asked to complete a questionnaire with the following background information, which we believed could affect the quality of the class diagrams:

- Number of months of relevant working experience.
- Credits in object-oriented programming.
- Proficiency in the use of the Tau UML Suite.

### 3.4 Assignment

There were two independent variables in this experiment: *process* (use case driven or validation) and *tool* (Tau UML Suite or pen and paper). Process and tool were assigned randomly. Table 1 shows the number of subjects in each group. The design is uneven because five of the subjects who had registered for the experiment, and filled in the questionnaire with background information, did not present themselves for the experiment.

**Table 1. Distribution of subjects**

| Process\Tool | Pen and paper | Tau | Total |
|---|---|---|---|
| **Use case driven** | 14 | 15 | 29 |
| **Validation** | 12 | 12 | 24 |
| **Total** | 26 | 27 | 53 |

### 3.5 Experimental Material

The task of the experiment was to construct a class diagram for a simple library system. This experiment is described in many books on UML, for example [18,24]. It was chosen because it is a well-known domain and simple enough for students just introduced to UML. The subjects received a textual requirements document and a use case model with the following use cases:

1. Borrow an item.
2. Hand in an item.
3. Check the status of an item.

The use cases were described using a template format based on those given in [9]. The subjects were given detailed guidelines on the two processes to follow. The processes were simple due to the size of the task and the time constraints on the experiment. The guidelines are shown in Table 2.

### 3.6 Procedure

The 53 subjects were present in the same laboratory. The subjects worked until they were finished, from 2.5 to 4.5 hours. In addition to the authors of this paper, two more persons were present during the experiment to help the subjects, with both understanding the experiment and the tools.

All the subjects used a Web-based tool for experiment support, the Simula Experiment Support Environment (SESE) [1], which provided the following functionality:

**Table 2**. **The exercise guidelines**

| Guidelines for the use case driven process | Guidelines for the validation process |
|---|---|
| **Exercise 1: Domain model**<br>1. Underline each noun phrase in the use case descriptions. Decide for each noun phrase whether it is a concept that should be represented by a class candidate in the domain model.<br>2. For the noun phrases that do not represent class candidates, decide whether these concepts should be represented as attributes in a domain model instead. (Not all attributes are necessarily found this way.) | **Exercise 1: Class diagram**<br>1. Underline all noun phrases in the requirements document. Decide for each noun phrase whether it is a concept that should be represented by a class in the class diagram.<br>2. For the noun phrases that do not represent classes, decide whether these concepts should be represented as attributes in the class diagram instead. (Not all attributes are necessarily found this way.)<br>3. Find the verbs or other sentences that represent actions performed by the system or system classes. Decide whether these actions should be represented by one ore more methods in the class diagram. (Not all methods needed are necessarily identified this way.) |
| **Exercise 2: Sequence diagrams**<br>1. Create one sequence diagram for each use case.<br>2. Study each use case description carefully, and underline the verbs or sentences describing an action. Decide for each action whether it should be represented by one or more methods in the sequence diagrams.<br>(Note! Not all methods needed are necessarily identified this way) | **Exercise 2: Sequence diagrams**<br>1. Create one sequence diagram for each use case.<br>2. Study each use case description carefully, and underline the verbs or sentences describing an action. Decide for each action whether it should be represented by one or more methods in the sequence diagrams.<br>(Note! Not all methods needed are necessarily identified this way) |
| **Exercise 3: Class diagram**<br>1. Transfer the domain model from exercise 1 into a class diagram.<br>2. Use the three sequence diagrams from exercise 2 to identify methods and associations. For each method in the sequence diagram:<br>o If an object of class *A* receives a method call *M*, the class *A* should contain the method *M* in the class diagram.<br>o If an object of class *A* calls a method of class *B*, there should be an association between the classes *A* and *B*. | **Exercise 3: Validation of the class diagram**<br>1. Consider each method in the sequence diagram. If several methods together form a system service, treat them as one service.<br>2. For each method or service:<br>o Confirm that the class that receives the method call contains the same or matching functionality.<br>o If an object of class A calls a method of class B, there should be an association between the classes A and B in the class diagram. If the class diagram contains any hierarchies, remember that it may be necessary to trace the hierarchy upwards when validating it.<br>If the validation in the previous steps failed, make the necessary updates. |

- A “pre-experiment” questionnaire to collect background information about the subjects was given to them before the start of the experiment.
- The experimental material was distributed through SESE.
- The solution documents from the subjects using the Tau UML Suite were uploaded to SESE when they were completed.
- Effort on each task was recorded by SESE.
- A “post-experiment” question about how well the subjects felt they had performed was given to the students immediately after they had finished their last task.

SESE also includes a think-aloud screen that pops up at pre-specified intervals [13]. The subjects used the think-aloud screen to comment on what they were doing every 15 minutes during the experiment. These comments enabled us to

- check whether the subjects actually followed the process descriptions,
- adjust the time recorded in cases where the subjects had particular problems or took breaks, and
- understand the solutions better.

# 4 Results and Analysis

This section describes the results and analysis of the experiment. The class diagrams were analysed according to the evaluation scheme presented in Section 3.2. This analysis was performed by a consultant who was not involved in the design or conduct of the experiment. He was also not involved in the teaching of the course.

## 4.1 Process Conformance

The think-aloud comments were considered together with the actual solutions to determine for each subject whether the given process was actually followed. We found that six of the subjects had major problems during the experiment; some had misunderstood the experiment, while others had too little knowledge of UML to perform the required tasks. One example of a person who was removed made the following comment when he was at the first task of the use case driven approach and therefore was supposed to make a domain model: “I am supposed to make a class diagram, not a domain model.” These six subjects were removed from the analysis. Three of them had been assigned to the use case driven process, while the other three had been assigned to the validation process. We did not, therefore find any difference in process conformance between the two processes.

## 4.2 Assessment of the Hypotheses

The Kruskal-Wallis statistical test was performed on the results (the scores on quality were ordinal and the data distributions were non-normal). A p-value of 0.05 was chosen as the level of significance. The results of the statistical tests are shown in

**Table 3. Comparing the two processes**

| Hypothesis | Process | N | Median | P-value | Reject |
|---|---|---|---|---|---|
| **$H1_0$ – Completeness** | Use case driven | 26 | 3.0 | | |
| | Validation | 21 | 3.0 | 0.016 | Yes |
| **$H2_0$ – Structure** | Use case driven | 26 | 4.0 | | |
| | Validation | 21 | 2.0 | 0.031 | Yes |
| **$H3_0$ – Time** | Use case driven | 6 | 188.0 | | |
| | Validation | 6 | 191.5 | 1.0 | No |

Table 3. The validation process gave a significantly higher score on completeness than did the use case driven process, that is, hypothesis $H1_0$ is rejected. The use case driven process gave significantly better structure than the validation process, so hypothesis $H2_0$ is also rejected.

Time was compared only for the subjects who produced good solutions, that is, those who obtained a score of 4 or 5 on completeness. For these subjects, the think-aloud comments were examined to see whether any of them had spent time on particular problems or on taking breaks. Based on these comments, the time recorded automatically by SESE was adjusted for some of the subjects. An example of a comment that was used to extract time for one of the subjects is the following: "Have spent approximately 10 min. on transferring some files because I was logged on with the wrong user ID, disregard this time". We found no difference in time spent between the two processes, so Hypothesis $H3_0$ was not rejected.

### 4.3 Influence of Background

In our opinion, the applicability of a specific method will depend on characteristics of the development project and of the development team. Therefore, some background information was collected about the subjects as described in Section 3.3. Stepwise regression was used to determine the extent to which the subjects' background had influenced the results.

A stepwise regression for the score on completeness, which included credits in programming, working experience and proficiency in the Tau UML Suite as well as the process, gave a model with R-square = 22.59. Process and number of credits were included in the model; process had higher significance than number of credits.

A corresponding model for the score on structure included all four factors and yielded a slightly higher R-square (25.53), this time with work experience as the most significant factor with process in second place.

The low R-squares of the two models show that these factors only explain a limited part of the results on quality. When attempting to construct a model to explain time spent, none of the factors were significant. It should, however, be noted that only a few of the subjects (11) had relevant working experience, and the subjects' background with regards to credits and proficiency in the use of the modelling tool was homogenous. A more heterogeneous group of subjects, with greater relevant working experience, may therefore have given different results.

Although we have experienced this before, it still attracts our attention that there was no relation between how the subjects assessed the quality of their own solution

and our assessment of quality. One explanation may be that the subjects had limited experience with UML, and had never participated in such an experiment before. Therefore, they had difficulty in determining what kind of solution was expected, and their assessment of their own work after the experiment was instead determined by their general self-confidence and ability to cope with the experimental setting.

# 5 Experiences from using a Professional Modelling Tool

One of the purposes of this study was to gain experience of conducting controlled experiments with a commercially available modelling tool. This section compares the results from those who used the Tau UML Suite with the results from those who used pen and paper. Experiences from organising such an experiment are also discussed.

## 5.1 Comparison of Results

Table 4 compares the results from those who used the Tau UML Suite with those who used pen and paper. The quality of the class diagrams, in terms of completeness and structure, was the same for both groups. However, those who used the Tau UML Suite spent more time, although not significantly more, on obtaining the same quality.

**Table 4. Comparing tool with pen and paper**

| Hypothesis | Tool | N | Median | P-value |
|---|---|---|---|---|
| **H1 – Completeness** | Pen and paper | 21 | 3.0 | |
| | Tau | 26 | 3.0 | 0.946 |
| **H2 – Structure** | Pen and paper | 21 | 3.0 | |
| | Tau | 26 | 3.0 | 0.947 |
| **H3 – Time** | Pen and paper | 8 | 172.0 | |
| | Tau | 8 | 212.5 | 0.247 |

We believe that even though the subjects were familiar with the tool, those who used the Tau UML Suite probably spent some extra time on understanding how to perform the tasks with it. They may also have been hindered by some minor bugs in the tool, and by the fact that it was very slow in some periods due to the heavy load of having 27 people working simultaneously. Those who used the tool probably also spent more time on getting the syntax correct to avoid error messages. After the experiment, the subjects who used pen and paper felt more confident about their own results than did those who used the tool.

However, more of the subjects who used pen and paper did not manage to complete the experiment; five as opposed to only one of those using the Tau UML Suite. In our opinion, this might be because those who used pen and paper were less motivated, because they found the experiment less realistic than did those who used the tool.

## 5.2 Assessment of the Hypotheses by Group

Table 5 compares the two processes for each of the two groups; those who used the Tau UML Suite and those who used pen and paper. The results obtained by those who used pen and paper were in agreement with the overall results; the validation process gave a significantly higher score on completeness than did the use case driven process, while the use case driven process resulted in a significantly better structure than did the validation process. For those who used the Tau UML Suite, however, there were no significant differences between the two processes. This shows that tool was an interacting factor in the experiment.

**Table 5. Comparing the processes by subgroups**

| Hypothesis | Group | Process | N | Median | P-value |
|---|---|---|---|---|---|
| **H1 – Completeness** | Pen and paper | Use case driven | 12 | 2.5 | |
| | | Validation | 9 | 5.0 | 0.011 |
| **H1 – Completeness** | Tau | Use case driven | 14 | 3.0 | |
| | | Validation | 12 | 3.0 | 0.436 |
| **H2 – Structure** | Pen and paper | Use case driven | 12 | 4.0 | |
| | | Validation | 9 | 2.5 | 0.046 |
| **H2 – Structure** | Tau | Use case driven | 14 | 4.0 | |
| | | Validation | 12 | 2.0 | 0.196 |

In our opinion, one reason for the different results in this experiment may be that those who used the Tau UML Suite did not follow the process descriptions as closely as did those who used pen and paper. Both process descriptions said that the sequence diagrams should be used actively in the construction of the final class diagram, but with the Tau UML Suite it is difficult to view several diagrams at the same time.

Further studies are, however, needed to investigate the differences between modelling using pen and paper and using a modelling tool. In such studies it would probably be useful to monitor in greater detail how the subjects actually work, for example by reducing the time interval between each think-aloud comment.

The subjects who used Tau UML Suite seemed to experience more problems during the experiment. On the other hand, it seems that the use of the tool led them to take the experiment more seriously, resulting in a quality similar to that obtained by the other group. In a larger experiment, however, the benefits of using a modelling tool would probably become more noticeable.

## 5.3 Experiences from Organising the Experiment

A principal motivation for using the Tau UML Suite was to gain experience of conducting experiments with professional tools, because it is our opinion that traditional pen-and-paper based exercises are hardly realistic for dealing with relevant problems of the size and complexity of most contemporary software systems. We have experienced that it is a challenge to configure an experimental environment with an infrastructure of supporting technology (processes, methods, tools, etc.) that resembles an industrial development environment. Our experience from replicating

several experiments with the use of professional tools is that using system development tools requires proper preparation [23]:

- Licences, installations, access rights, etc. must be checked.
- The subjects must be or become familiar with the tools.
- The tools must be checked to demonstrate acceptable performance and stability when many subjects are working simultaneously.

# 6 Threats to Validity

This section presents and discusses threats to the validity of our results.

## 6.1 Measuring Quality

In this experiment we attempted to measure the quality of class diagrams. The quality in terms of completeness is subjective, but the domain of the experiment was in this case based on a well-known example from textbooks. Moreover, three persons were involved in determining what would be a correct solution (one of them an external consultant who was not involved in the experiment otherwise).

Well-defined metrics for measuring coupling and cohesion exist, for example, those described in [6,7], but since the class diagrams produced in this experiment were quite small and simple, these metrics were not easily applicable. Therefore, coupling and cohesion were also measured subjectively.

It is difficult to define, and therefore to measure, the quality of a class diagram. The use of a combination of several different independent measures of quality would improve our evaluation scheme. In addition to assessing correctness we could, for example, assess syntactic correctness or generate code from the class diagrams and evaluate the code. Several independent evaluators would also represent an improvement.

## 6.2 Realism of the Experimental Design

The subjects were novices to modelling with UML. Therefore, conducting the experiment with more experienced subjects might have led to different results. However, we believe that our subjects are representative for developers with little experience with UML, and these may also be most in need of guidance from a defined process. Nevertheless, we intend to conduct the experiment with professional software developers as subjects.

Another threat to validity is that the procedure of the experiment differed in several ways from the way in which software developers actually work when designing with UML: The subjects spent from 2.5 to 4.5 hours on designing a class model, which is much shorter than a typical design process in an industrial setting. In addition, the subjects worked individually, while design is typically done in teams. An experimental setting will always differ to some extent from actual practice, as the

subjects may have found the setting stressful, and we know from the think-aloud comments that a few of the subjects were disturbed or stressed during the experiment.

## 7 Conclusions and Future Work

Several approaches have been proposed for the transition from functional requirements to a design model, but these approaches have been subject to little empirical validation. A use case driven development process, in which the use case model is the principal basis for a class diagram, is recommended together with the UML, but a number of problems with this process have been reported. An alternative process, a validation process, where a use case model is applied in validating the design, has therefore been proposed.

We conducted an experiment with 53 students as subjects to compare a use case driven process with a validation process. The aim of the experiment was to investigate differences between the two approaches with respect to the quality of the resulting class diagrams in terms of completeness and structure, and with regards to differences in time spent on obtaining a good design.

The results show that the validation process resulted in class diagrams that implemented significantly more of the requirements, but also that the use case driven process resulted in class diagrams with a significantly better structure than did the validation process. There was no difference in time spent between the two processes.

The results confirm the results from a pilot experiment that we conducted previously. In our opinion, the results support the claims that a use case model is insufficient for deriving all necessary classes and may lead the developers to mistake requirements for design. We also believe that, based on these results, it may be beneficial to apply a use case driven process when the use case model contains many details and there is a strong need for good structure, but apply the use case model in validation otherwise.

Half of the subjects in the experiment used the modelling tool Tau UML Suite from Telelogic in the design of class diagrams; the other half used pen and paper. Our experience is that the use of a professional modelling tool in an experiment requires much more effort and support both from subjects and organisers, but is more realistic and may also be more motivating for the subjects.

One experiment can only provide insight on how the alternative processes perform in a limited context. Therefore, to gain knowledge that is transferable to actual software development practice, it will be necessary to conduct a series of experiments in different environments. An experiment permits the in-depth study of some aspects of a development process, but an experimental context will necessarily differ from actual work practice, so experiments should be combined with other types of studies, for example, case studies. We intend to conduct further studies to investigate how to apply a use case model in an object-oriented design process. In particular, we intend to

- improve the measuring of quality of the resulting class diagrams by combining several aspects and have the analysis done by several independent evaluators,

- increase the realism of the experiment by using professionals as subjects, letting them work in teams instead of as individuals and also increase the size and complexity of the task,
- improve the collection of background data, as well as process information during the experiment, to study which process attributes and skills actually affect the quality of the object-oriented design, and
- extend the evaluation to include some of the other approaches to designing a class diagram that were briefly described in this report.

## Acknowledgements

We acknowledge all the students at the University of Oslo who participated in the experiment. We also acknowledge Eskild Bush for support on the use of the Tau UML Suite, Gunnar J. Carelius for adapting the experiment for use with the Simula Experiment Support Environment and for support during the experiment, Per Thomas Jahr for analysing the class diagrams, Terje Knudsen for technical support as well as Tanja Grutshke, Christian Herzog, Tor Even Ramberg and Sinan Tanilkan for debugging the experimental material.

## References

1. Arisholm, E., Sjøberg, D., Carelius, G.J. and Lindsjørn, Y. A Web-based Support Environment for Software Engineering Experiments. Nordic Journal of Computing, Vol. 9, pp. 231-247, 2002.
2. Arlow, J. and Neustadt I. UML and the Unified Process. Practical Object-Oriented Analysis and Design. Addison-Wesley, 2002.
3. Batra, D., Hoffer, J.A. and Bostrom, R.P. Comparing representations with relational and EER models. Communications of the ACM, 33(2), pp. 126-139, 1990.
4. Bennett, S., McRobb, S. and Farmer, R. Object-Oriented Systems Analysis and Design using UML. McGraw-Hill, New York, 1999.
5. Booch, G., Rumbaugh, J. and Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley, 1999.
6. Briand, L.C., Daly, J. and Wüst, J. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering, 3(1), pp. 65-117, 1998.
7. Briand, L.C, Daly, J.W. and Wüst, J. A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on Software Engineering, 25(1), pp. 91-121, 1999.
8. Briand, L.C., Labiche, Y. and Soccar, G. Automating Impact Analysis and Regression Test Selection Based on UML Designs. Technical Report, Carleton University, SCE-02-04, 2002.
9. Cockburn, A. Writing Effective Use Cases. Addison-Wesley, 2000.
10. Harrison, W. N=1: An Alternative for Software Engineering Research? Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Workshop, 5 June, 2000 at 22nd Int. Conf. on Softw. Eng. (ICSE), Limerick, Ireland, pp. 39-44, 2000.

11. Jacobson, I., Christerson, M., Jonsson P. and Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.
12. Jacobson, I., Booch, G., and Rumbaugh, J. The Unified Software Development Process. Addison-Wesley, 1999.
13. Karahasanovic, A., Anda, B., Jørgensen, M. and Sjøberg, D.I.K. Tool Support for the Think-Aloud Protocol. Paper in preparation.
14. Larman, C. Applying UML and Patterns—an Introduction to Object-Oriented Analysis and Design and the Unified Process. Prentice-Hall, Englewood Cliffs, NJ, 2002.
15. Lethbridge, T.C. and Laganiere, R. Object-Oriented Software Engineering: Practical Software Development Using UML and Java. McGraw-Hill, New York, 2001.
16. Liang, Y. From use cases to classes: a way of building object model with UML. Information and Software Technology, 45(2), pp. 83-93, 2003.
17. Maciaszek, L. Requirements Design and System Analysis. Addison-Wesley 2001.
18. Richter, C. Designing Flexible Object-Oriented Systems with UML. Macmillan Technical Publishing, 1999.
19. Rosenberg, D. and Scott, K. Applying Use Case Driven Object Modeling with UML. An Annotated E-commerce Example. Addison-Wesley, 2001.
20. Sharble, R.C. and Cohen, S.S. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. Software Engineering Notes, 18(2), pp. 60-73. 1993.
21. Shull, F., Travassos, G., Carver, J. & Basili, V. Evolving a Set of Techniques for OO Inspections. University of Maryland Technical Report CS-TR-4070, October 1999.
22. Sjøberg, D.I.K, Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F. and Vokac, M. Conducting Realistic Experiments in Software Engineering. ISESE'2002 (First International Symposium on Empirical Software Engineering), Nara, Japan, October 3-4, pp. 17-26, IEEE Computer Society, 2002.
23. Sjøberg, D., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A. and Vokác, M. Challenges and Recommendations when Increasing the Realism of Controlled Software Engineering Experiments, Submitted for publication, 2003.
24. Stevens, P. and Pooley, R. Using UML. Software Engineering with Objects and Components. Addison-Wesley, 2000.
25. Syversen, E., Anda, B. and Sjøberg, D.I.K. An Evaluation of Applying Use Cases to Construct Design versus Validate Design, Hawaii International Conference on System Sciences (HICSS-36), Big Island, Hawaii, January 6-9, 2003.
26. Telelogic Tau UML Suite, www.telelogic.com.
27. The UML meta-model, version 1.4. www.omg.org, 2002.
28. Wirfs-Brock, R., Wilkerson, B. and Wiener, L. Designing Object-Oriented Software. Prentice Hall, 1990.

# Bibliography


Abdel-Hamid, T.K. and Madnick, S.E. Lessons Learned from Modeling the Dynamics of Software Development. *Communication of the ACM*, 32(12), pp. 1426-1438, 1989.

Adelson, B. Problem solving and the development of abstract categories in programming languages. *Memory and Cognition*, 9(4), pp. 422-433, 1981.

Adrion, W.R. Research Methodology in Software Engineering. *ACM Software Engineering Notes*, 18(1), pp. 36-37, 1993.

Albrecht, A.J. Measuring Application Development Productivity. Proceedings of Joint SHARE, GUIDE, and IBM Application Development Symposium, 1979.

Anderson, R.C., and Pearson, P.D. *A schema theoretic view of basic processes in reading*. P.D. Pearson (Ed.) Handbook on reading research. New York, Longman, 1984.

Arisholm, E., Anda, B., Jørgensen, M. and Sjøberg, D. Guidelines on Conducting Software Process Improvement Studies in Industry. 22nd IRIS Conference (Information Systems Research Seminar in Scandinavia), Keuruu, Finland, pp. 87-102, 1999.

Arisholm, E., Sjøberg, D., Carelius, G.J. and Lindsjørn, Y. A Web-based Support Environment for Software Engineering Experiments. *Nordic Journal of Computing*, Vol. 9, pp. 231-247, 2002.

Arisholm, E., Sjøberg, D. and Jørgensen, M. Assessing the Changeability of two Object-Oriented Design Alternatives – a Controlled Experiment, *Empirical Software Engineering*, 6(3), pp. 231-277, September 2001.

Arlow, J. and Neustadt I. *UML and the Unified Process. Practical Object-Oriented Analysis and Design*. Addison-Wesley, 2002.

Armour, F. and Miller, G. *Advanced Use Case Modelling*. Addison-Wesley, 2000.

Arnold, P. and Pedross, P. Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department. Forging New Links. IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 490-493, 1998.

Bartlett, F. C. *Remembering. A Study in Experimental and Social Psychology*. Cambridge University Press, 1932.

Basili, V.R. The Role of Experimentation in Software Engineering: Past, Current, and Future, Proceeding of the 18th International Conference on Software Engineering, Berlin, Germany, March 25-29, pp. 442-449, 1996.

Basili, V. R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. and Zelkowitz, M. The Empirical Investigation of Perspective-Based Reading. *Empirical Software Engineering Journal*, 1(2), pp. 133-146, 1996.

Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S. and Zelkowitz, M. Lab Package for the Empirical Investigation of Perspective.Based Reading, 1998. At: http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html.

Basili, V.R., Rombach, D. and Selby, R. The Experimental Paradigm in Software Engineering. Experimental Engineering Issues: Critical Assessment and Future

Directions, International Workshop, Dagstuhl, Germany, Springer-Verlag, LNCS, No. 706, 1993.

Basili, V.R., Selby, R. and Hutchens, D. Experimentation in Software Engineering. *IEEE Transactions on Software Engineering* (invited paper), July 1986.

Basili, V.R., Shull, F.and Lanubile, F. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25(4), pp. 456-473, 1999.

Batra, D., Hoffer, J.A. and Bostrom, R.P. Comparing representations with relational and EER models. *Communications of the ACM*, 33(2), pp. 126-139, 1990.

Ben Achour, C., Rolland, C., Maiden, N.A.M. & Souveyet, C. Guiding Use Case Authoring: Results of an Empirical Study. Proceedings IEEE Symposium on Requirements Engineering, IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.

Bennett, S., McRobb, S. and Farmer, R. *Object-Oriented Systems Analysis and Design using UML*. McGraw-Hill, New York, 1999.

Biddle, R., Noble, J. and Tempero, E. Supporting reusable use cases. In Gacek, C. (Ed.) Proceedings of ICSR'7 – 7th International Conference on Software Reuse: Methods, Techniques, and Tools, pp. 210-226, LNCS'2319, Springer Verlag, 2002.

Blattberg, R.C. and Hoch, S.J. Database models and managerial intuition: 50% model + 50% manager, *Management Science,* 36(8), pp. 887-899, 1990.

Boehm, B.W. *Software Engineering Economics*. Prentice-Hall, 1981.

Boehm, B. and Port, D. Escaping the Software Tar Pit: Model Clashes and How to Avoid Them. *Software Engineering Notes*, 24(1), 36-48, 1999.

Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

Briand, L., Arisholm, E., Counsell, S., Houdek, F. and Thévenod-Fosse, P. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions, *Empirical Software Engineering*, 4(4), pp. 387-404, 1999a.

Briand, L.C., Daly, J. and Wüst, J. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering*, 3(1), pp. 65-117, 1998a.

Briand, L.C, Daly, J.W. and Wüst, J. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 25(1), pp. 91-121, 1999b.

Briand, L., El Emam, K. Fussbroich, T., Laitenberger, O., Using Simulation to build inspection efficiency benchmarks for development projects. Proceedings of the 20th International Conference on Software Engineering. IEEE Computer Society Press, Silver Spring, MD, pp. 340-349, 1998b.

Briand, L.C, Labiche, Y. A UML-based Approach to System Testing. In M. Gogolla and C. Kobryn (Eds.): UML 2001 – 4th International Conference on the Unified Modeling Language, Toronto, Canada, October 1-5, 2001, pp. 194-208, LNCS 2185, Springer-Verlag, 2001.

Briand, L.C., Labiche, Y. and Soccar, G. Automating Impact Analysis and Regression Test Selection Based on UML Designs. Technical Report, Carleton University, SCE-02-04, 2002.

Briand, L.C. and Wüst, J. The Impact of Design Properties on Development Cost in Object-Oriented Systems. ISERN Technical Report TR-99-16, 1999.

Britton, C. and Jones, S. The untrained eye: how languages for software specification support understanding in untrained users. *Human-Computer-Interaction*, 14(1-2), pp. 191-244, 1999.

Brooks, R. Towards a Theory for the Comprehension of Computer Programs. *International Journal of Human-Computer Studies*, 51(2), pp. 197-211, 1999.

Burkhardt, J.-M., Détienne, F. and Wiedenbeck, S. Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase. *Empirical Software Engineering*, 7(2), pp. 115-156, 2002.

Cheng, B. and Jeffery, R. Comparing Inspection Strategies for Software Requirement Specifications. Proceedings Australian Software Engineering Conference. IEEE Comput. Soc, Los Alamitos, CA, USA, 1996.

Cioch, F.A. Measuring Software Misinterpretation. *Journal of Systems and Software*, 14(2), pp. 85-95, February 1991.

Ciolkowski, M. ISERN Inspection Experiment. Presentation at ISERN 2002 Annual Meeting, Nara, Japan, 30 September 2002.

Cockburn, A. Structuring Use Cases with Goals. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, Ha.T.TR.95.1, http://members.aol.com/ acockburn/papers/usecases.htm, 1995.

Cockburn, A. *Writing Effective Use Cases*. Addison-Wesley, 2000.

Constantine, L. L. & Lockwood, L. A. D. *Software for Use. A Practical Guide to the Models and Methods for Usage-Centered Design*. Addison-Wesley, 1999.

Cox, K. & Phalp, K. Replicating the CREWS Use Case Authoring Guidelines. *Empirical Software Engineering Journal*, 5(3), pp. 245-268, 2000.

Cox, K. and Phalp, K. Supporting Communicability with Use-Case Guidelines: An Empirical Study. EASE 2002 – Empirical Assessment in Software Engineering, Keele, UK, April 8-10, 2002.

Cunningham, J.B. Case study principles for different types of cases. *Quality and Quantity*, 31, pp. 401-423. 1997.

Deligiannis, I.S, Shepperd, M., Webster, S. and Roumelitos, M. A Review of Experimental Investigations into Object-Oriented Technology. *Empirical Software Engineering*, 7(3), pp. 193-231, 2002.

Detienne, F. Une application de la théorie des schémas a la comprehension de programmes. *Le Travail Humain*, 51(4), pp. 335-350, 1988.

Doolan, E.P. Experience with Fagan's inspection method. *Software Practice and Experience*, 22(2), pp. 173-182, February 1992.

Dybå, T. *Enabling Software Process Improvement: An Investigation of the Importance of Organizational Issues*, Doctoral Dissertation, Norwegian University of Science and Technology, 2001:101, ISBN 82-471-5371-8, 2001.

El Emam, K. and Laitenberger, O. Evaluating Capture-Recapture Models with Two Inspectors. *IEEE Transactions on Software Engineering*, 27(9), pp. 851-864, September 2001.

Fagan, M.E. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3), pp. 182-211, 1976.

Fetcke. T., Abran, A. and Nguyen, T.-H. Mapping the OO-Jacobson Approach into Function Point Analysis. International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-23). IEEE Comput. Soc, Los Alamitos, CA, USA, pp. 192-202, 1998.

Firesmith, D.G. Use Case Modeling Guidelines. Proceedings of Technology of Object-Oriented Languages and Systems – TOOLS 30. IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.

Flyvbjerg, M. *Rationalitet og Magt*. Akademisk forlag, 1991.

Gilb, T. and Graham, D. *Software Inspection*. Addison-Wesley, 1993.

Gilmore, D. J. and Green, T.R.G. Comprehension and recall of miniature programs. *Journal of Man-Machine Studies,* Vol. 21, pp. 31-48, 1984.

Glasgow, J., Narayanan, N.H. and Chandrasekaran, B. *Diagrammatic reasoning : cognitive and computational perspectives*. Menlo Park, Calif.: AAAI Press, 1995.

Glass, R.L. The Software Research Crisis. *IEEE Software*, 11(6), pp. 42-47, 1994.

Glass, R. *Software Runaways*, Harlow, Prentice Hall, 1998.

Harrison, W. N=1: An Alternative for Software Engineering Research? Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Workshop, 5 June, 2000 at 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, pp. 39-44, 2000.

Harwood, R. J. Use case formats: Requirements, analysis, and design. *Journal of Object-Oriented Programming*, 9(8), pp. 54-57, January 1997.

Hooper, J.W. and Hsia, P. Scenario-based Prototyping for Requirements Identification. *ACM Sigsoft Software Engineering Notes*, 7(5), pp. 88-93, 1982.

Hufnagel, E.M. and Conca, C. User response data: The potential for errors and biases. *Information Systems Research*, 5(1), pp. 48-73, 1994.

Hurlbut, R.R. A Survey of Approaches for Describing and Formalizing Use Cases. Technical Report: XPT-TR-97-03, Expertech, Ltd., 1997.

Høst, M., Regnell, B. and Wohlin, C. Using Students as Subjects – A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering Journal*, 5(3), pp. 201-214, 2000.

Jaaksi, A. Our Cases with Use Cases. *Journal of Object-Oriented Programming*, 10(9), pp. 58-64, February 1998.

Jacobson, I., Booch, G., and Rumbaugh, J. *The Unified Software Development Process*. Addison-Wesley, 1999.

Jacobson, I. et al. *Object-Oriented Software Engineering. A Use Case Driven Approach.* Addison-Wesley, 1992.

Jeffery, D.R., Low, G.C. and Barnes, M. A comparison of function point counting techniques. *IEEE Transactions on Software Engineering*, 19(5), pp. 529-532, 1993.

Jørgensen, M. An empirical evaluation of the MkII FPA estimation model. Norwegian Informatics Conference, Voss, Norway, 1997.

Jørgensen, M. A Review of Studies on Expert Estimation of Software Development Effort. To appear in *Journal of Systems and Software*, 2004.

Jørgensen, M., Kirkebøen, G., Sjøberg, D., Anda and B., Bratthall, L. Human judgement in effort estimation of software projects. Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Workshop, 5 June, 2000 at 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland, pp. 45-51, 2000.

Jørgensen, M. and Sjøberg, D.I.K. Impact of Software Effort Estimation on Software Work. *Information and Software Technology*, Vol. 43, pp. 939-948, 2001a.

Jørgensen, M. and Sjøberg, D.I.K. Software Process Improvement and Human Judgement Heuristics. *Scandinavian Journal of Information Systems*, Vol. 13, pp. 99-122, 2001b.

Karahasanovic, A., Anda, B., Jørgensen, M. and Sjøberg, D.I.K. Tool Support for the Think-Aloud Protocol. Paper in preparation, 2003.

Karner, G. Metrics for Objectory. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21, December 1993.

Kautz-K. and Pries-Heje, J. Systems development education and methodology adoption. *Computer Personnel*, 20(3), pp. 6-26, July 1999.

Kemerer, F.K. Reliability of Function Points Measurement. *Communications of the ACM*. 36, (2), pp. 85-97, February 1993.

Kitchenham, B. *Software metrics: measurement for software process improvement*. Blackwell Publishers, 1996.

Kitchenham, B.A.; Pfleeger, S.L.; Pickard, L.M.; Jones, P.W.; Hoaglin, D.C.; El-Emam, K.; Rosenberg, J. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8), pp, 721-734, 2002.

Kitchenham, B., Pickard, L. and Pfleeger, S.L. Case Studies for Method and Tool Evaluation. *IEEE Software*, 12(4), pp. 52-62, 1995.

Kosslyn , S. M. Understanding charts and graphs. *Applied Cognitive Psychology*, Vol. 3, pp. 185-223, 1989.

Kulak, D. & Guiney, E. *Use Cases: Requirements in Context*. Addison-Wesley, 2000.

Laitenberger, O., Atkinson, C., Schlich, M. and El Emam, K. An experimental comparison of reading techniques for defect detection in UML design documents. *Journal of Systems and Software*, 53(2), pp. 183-204, August 2000.

Lanubile.F. and Visaggio, G. Assessing defect detection methods for software requirements inspections through external replication, ISERN-96-01, January 1996.

Larman, C. Applying UML and Patterns—an Introduction to Object-Oriented Analysis and Design and the Unified Process. Prentice-Hall, Englewood Cliffs, NJ, 2002.

Lethbridge, T.C. and Laganiere, R. *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*. McGraw-Hill, New York, 2001.

Letovsky, S. and Soloway, E. Delocalized plans and program comprehension. IEEE Software, 3(4), pp. 41-49, 1986.

Liang, Y. From use cases to classes: a way of building object model with UML. *Information and Software Technology*, 45(2), pp. 83-93, 2003.

Linos, P. *et al.* Facilitating comprehension of C-programs: an experimental study. Proceedings IEEE Second Workshop on Program Comprehension, pp. 55-63, 1993.

Liwu, L. Use case patterns. *International Journal of Software Engineering and Knowledge Engineering* 12(1), pp. 19-40, Feb. 2002.

Maciaszek, L. *Requirements Design and System Analysis*. Addison-Wesley 2001.

Marchesi, M. OOA Metrics for the Unified Modeling Language. In Proc. of the Second Euromicro Conference on Software Maintenance and Reengineering, IEEE Comput. Soc, Los Alamitos, CA, USA; pp. 67-73, 1998.

Martinsen, S.A. & Groven, A-K. Improving Estimation and Requirements Management Experiences from a very small Norwegian Enterprise. SPI 98 Improvement in Practice: Reviewing Experience, Previewing Future Trends. The European Conference on Software Improvement. Meeting Management, Farnham, UK, 1998.

Mattingly, L. & Rao, H. Writing Effective Use Cases and Introducing Collaboration Cases. *Journal of Object-Oriented Programming*, 11(6), pp. 77-79, 81-84, 87, October 1998.

Miara, R. *et al.* Program indentation and comprehensibility. *Communications of the ACM*, vol. 26, pp. 861-867, 1983.

Miller, J., Wood, M., Roper, M. and Brooks, A. Further Experiences with Scenarios and Checklists. *Empirical Software Engineering*, 3(1), pp. 37-64, January 1998.

Næss, Arne. *Vitenskapsfilosofi*. 3rd ed. Universitetsforlaget. Oslo, 1980.

Pfleeger, S.L. Albert Einstein and Software Engineering. *IEEE Computer*, pp. 32-37, October 1999.

Pfleeger, S.L. Experimental Design and Analysis in Software Engineering. Part 2: How to Set Up an Experiment. *ACM Software Engineering Notes*, 20(1), pp. 22-26, 1995.

Pfleeger, S. L. *Software Engineering. Theory and Practice*. Prentice Hall Inc., 1998.

PITAC – President's Information Technology Advisory Committee Report to the President, at www.itrd.gov/ac/report/, February 24, 1999

Pressley, M. & McCormick, C. B. *Advanced Educational Psychology for Educators, Researchers and Policymakers*. Harper Collins, 1995.

Porter, A.A., Votta, L.G. and Basili, V.R. Comparing Detection Methods for Software Requirements Inspections: a Replicated Experiment. *IEEE Transactions on Software Engineering*, 21(6), pp. 563-575, June 1995.

Regnell, B. Requirements Engineering with Use Cases – A Basis for Software Engineering. PhD Thesis, Lund University, 1999a.

Regnell, B., Andersson, M. & Bergstrand, J. A Hierarchical Use Case Model with Graphical Representation. Proceedings of Second IEEE International Symposium on Requirements Engineering (RE'95), York, UK, 1995.

Regnell, B., Runeson, P. and Thelin T. Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements, Technical Report CODEN: LUTEDX(TETS-7172)/1-40/1999, Dept. of Communication Systems, Lund University, 1999b.

Ribu, K. Estimating Object-Oriented Software Projects with Use Cases. Masters Thesis, University of Oslo, November 2001.

Richter, C. *Designing Flexible Object-Oriented Systems with UML*. Macmillan Technical Publishing, 1999.

Rombach et al. Experimental Software Engineering Issues: critical Assessment and Future Directions, Dagstuhl Workshop, Germany, September, LNCS 706, Springer-Verlag. 1993.

Rosenberg, D. and Scott, K. *Use Case Driven Object Modelling with UML*. Addison-Wesley, 1999.

Rosenberg, D. and Scott, K. *Applying Use Case Driven Object Modeling with UML. An Annotated E-commerce Example*. Addison-Wesley, 2001.

Sandahl, K., Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M. and Ohlsson, N. An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections, *Empirical Software Engineering*, 3(4), pp. 327-354, December 1998.

Schneider, G. & Winters, J. *Applying Use Cases – A Practical Guide.* Addison-Wesley, 1998.

Schneiderman and Mayer. Syntactic/semantic interactions in programmer behaviour: a model and experimental results. *International Journal of Computer and Information Sciences*, Vol. 8, pp. 219-238, 1979.

Sendall, S. and Stroheimer, A. From Use Cases to System Operation Specification. Third International Conference on the Unified Modeling Language (UML'2000), York, UK. LNCS 1939; Springer Verlag, pp. 1-15, 2000.

Sharble, R.C. and Cohen, S.S. The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. *Software Engineering Notes*, 18(2), pp. 60-73, 1993.

Sheetz, S.D. Identifying the difficulties of object-oriented development. *Journal of Systems and Software*, 64(1), pp. 23-36, 2002.

Shull, F., Basili, V., Carver, J. Maldonado, J.C., Travassos, G.H., Mendonça, M. and Fabbri, S. Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem. ISESE'2002 (First International Symposium on Empirical Software Engineering), Nara, Japan, October 3-4, 2002, pp. 7-16, IEEE Computer Society, 2002.

Shull, F., Rus, I. and Basili, V. How Perspective-Based Reading Can Improve Requirements Inspections. *IEEE Computer*, 33(7), pp. 73-79, July 2000.

Shull, F., Travassos, G., Carver, J. & Basili, V. Evolving a Set of Techniques for OO Inspections. University of Maryland Technical Report CS-TR-4070, October 1999.

Sommerville, I. *Software Engineering*, 6th Edition, Addison-Wesley, 2000.

Sjøberg, D.I.K, Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F. and Vokac, M. Conducting Realistic Experiments in Software Engineering. ISESE'2002 (First International Symposium on Empirical Software Engineering), Nara, Japan, October 3-4, 2002, pp. 17-26, IEEE Computer Society, 2002.

Sjøberg, D., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A. and Vokác, M. Challenges and Recommendations when Increasing the Realism of Controlled Software Engineering Experiments, Submitted for publication, 2003.

Smith, J. The Estimation of Effort Based on Use Cases. Rational Software, White paper, 1999.

Soloway, E., Adelson, B. and Ehrlich, K. Knowledge and process in the comprehension of computer programs. In Chi, M.T.H. *et al*. (Eds) *The nature of expertise*. Lawrence Erlbaum Assoc., pp. 129-152, 1988.

Sparks, S. and Kaspcynski, K. The Art of Sizing Projects, Sun World. 1999.

Stensrud, E. Empirical Studies in Software Engineering Management of Package-Enabled Reengineering Project. PhD thesis, University of Oslo, 2000.

Stevens, P. and Pooley, R. Using UML. *Software Engineering with Objects and Components*. Addison-Wesley, 2000.

Symons C.R. *Software Sizing and Estimating, MKII FPA*. John Wiley and Sons, 1991.

Symons, C. Come back function point analysis (modernized) – all is forgiven! Software Measurement Services Ltd., 2001.

Syversen, E., Anda, B. and Sjøberg, D.I.K. An Evaluation of Applying Use Cases to Construct Design versus Validate Design, Hawaii International Conference on System Sciences (HICSS-36), Big Island, Hawaii, January 6-9, 2003.

Tassc:Estimator, www.tassc-solutions.com, 2002

Telelogic Tau UML Suite, www.telelogic.com, 2002.

The Object Factory. Estimating Software Projects using ObjectMetrix, White paper, April 2000.

The UML meta-model, version 1.4. www.omg.org, 2002.

Thelin, T., Runeson, P. And Wohlin, C. An Experimental Comparison of Usage-Based and Checklist-Based Reading. WISE 2001.

Tichy, W.F. Should Computer Scientists Experiment More? 16 Reasons to Avoid Experimentation, *IEEE Computer*, 31(5), pp. 32-40, May 1998.

Tichy, W.F. Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering*, 5(4), pp. 309-312, 2000.

Vidger, M.R. and Kark, A.W. Software Cost Estimation and Control. National Research Council Canada, Institute for Information Technology. NRC No. 37116, 1994.

Von Mayrhauser, A & Lang, S. "A Coding Scheme to Support Systematic Analysis of Software Comprehension". *IEEE Transactions on software Engineering*, 25(4), pp. 526-540, 1999.

Wallace, W.L. *Sociological theory*, Aldine, Chigaco, 1969.

Weidenhaupt, K., Pohl, K., Jarke, M. and Haumer, P. Scenarios in system development: current practice. *IEEE Software*, 15(2), pp. 34-45, March-April 1998.

Winn, W. An Account of How Readers Search for Information in Diagrams. *Contemporary Educational Psychology*, 18(2), pp. 162 - 185, 1993.

Wirfs-Brock, R., Wilkerson, B. and Wiener, L. *Designing Object-Oriented Software*. Prentice Hall, 1990.

Wohlin, C., Runeson, P., Høst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publisher, USA. 2000.

Wood, M., Daly, J., Miller, J. and Roper, M. Multi-method research: An empirical investigation of object-oriented technology. *Journal of Systems and Software*, 48(1), pp. 13-26, 1999.

Yourdon, E. *Structured Walkthroughs*. Prentice-Hall, 1989.

Zelkowitz, M.V. and Wallace, D.R. Experimental Models for Validating Technology. *IEEE Computer*, 31(5), pp. 23-31, May 1998.