UNIVERSITY OF OSLO

Department of informatics

A Systematic Review of Empirical Research on Model-Driven Development with UML

Master thesis

60 credits

Terese Helene Haug

February 2007

Abstract

Model-Driven Development (MDD) with UML is gaining widespread use in the IT-industry. However, little is known with regards to the actual costs and benefits of MDD with UML. This thesis is a systematic review of 21 selected articles reporting empirical studies on MDD with UML, published between 2000 and 2005, in three leading journals and one conference proceeding. The objective of the investigation is to get an overview of the state-of-the-art for empirical studies of MDD with UML, and show the typical information found in articles of this topic. The data collected during analysis of these 21 articles, was used to address the following issues: The amount of empirical research of MDD with UML, the extent of different empirical research methods used, what different UML-diagrams that are studied and their benefits, for which application domains UML are studied, to what extent UML is compared to other approaches, the possibilities for meta-analysis and what the authors suggest about future research.

The main conclusion of this study is that it does not exist sufficient empirical evidence to draw conclusions regarding the usefulness of Model-Driven Development with UML. The few existing empirical studies of MDD with UML do mostly suggest that UML is useful, but they are too few and deal with too many aspects of UML-usage, to allow for definite conclusions about the usefulness of the technique. Further, the main findings of this review are the following. Experiments is the most used research method within studies of MDD with UML, and this study found four times as many experiments as case studies. UML usage was found to yield benefits for the comprehension, construction and correctness of a system. However, the study also revealed that the benefits of UML-usage are often dependent on the application domain and the experience and abilities of developers. Furthermore, few studies exist that investigate the cost-effectiveness of UML-usage, and most studies have focus on detailed aspects of the use of single diagrams. Hence, the results could be useful for companies that already make extensive use of UML, but these results do not provide much guidance when UML is introduced in a company.

Acknowledgements

First of all, I would like to thank my supervisor, Bente Anda, for her guidance, encouragement and support with this work. I would also like to thank Erik Arisholm for his involvement and valuable contributions. Thanks to Sturle Helland for the cooperation when planning and performing this review. I am also grateful to Simula Research Laboratory for providing such great facilities and work environment. Last but not least, my gratitude goes to my family and friends for their encouragement and support during this period.

Oslo, February 2007

Terese Helene Haug

Contents

List of Tables

1. Introduction

Section 1.1 presents the motivation, and Section 1.2 presents the objective of the research and states the research questions that is investigated in this thesis. Section 1.3 describes the contributions of this work. The last Section of the introduction Section presents the structure of the remainder of this thesis.

1.1 Motivation

Model-Driven Development (MDD) with UML is gaining widespread use in the IT-industry, and aims to raise the level of abstraction for software development by the use of models as key artefacts in software development, from system specification and analysis, to design and testing. The use of UML is claimed among others to improve the quality of software product deliverables, to support reuse and reduce the effort of developing and maintaining the software product.

However, little is known with regards to the actual costs and benefits of MDD with UML. Briand et al. [8] state that many methods, processes, tools or notations are being used without thorough evaluation. Sjøberg et al. [19] write that research in empirical software engineering should aim to acquire general knowledge about which *technology* (process, method, technique, language, or tool) is useful for *whom* to conduct which (software engineering) *tasks* in which *environments*. Thus, there is a need for understanding different properties, advantages and drawbacks of MDD with UML. That is, when you should use the technique, to what extent and what benefits and costs it will entail.

A few surveys have been conducted to determine the state of Software Engineering research as a whole with respect to topic, research approach, research method, reference discipline and level of analysis [9, 11, 18, 19, 22]. To the authors' knowledge there has however not been performed any studies that thoroughly cover empirical studies of Model-Driven Development with UML.

Briand et al. [8] state that the overall objective of empirical studies of object-oriented technologies and products is to gather tangible evidence about its properties and gain deeper insights into the nature of the object-oriented paradigm and its relationship to other approaches.

Software Engineering is a relatively new research field and a strong experimental model of the field has not yet been developed. It is, however, more and more recognized that empirical studies need to be combined and conclusions need to be generalized in order to build a body of evidence to provide a scientific foundation for the engineering of software products. This is the field of meta-analysis. Miller [15] states reasons for performing meta-analytical procedures like this; "deriving reliable empirical results from a single experiment is an unlikely event. Hence to progress multiple experiments must be undertaken per hypothesis and the subsequent results effectively combined to produce a single reliable conclusion."

As the state-of-the art of Model-Driven Development with UML has not been thoroughly investigated, I focus on this topic in this thesis.

1.2 Objective

This thesis is a systematic review of 21 selected articles published between 2000 and 2005 in IEEE Transactions on Software Engineering, Empirical Software Engineering, the conference proceeding UML/MODELS and the Requirements Engineering journal, that report empirical studies on Model-Driven Development (MDD) with UML. The objective of the investigation is to get an overview of the state-of-the-art for empirical studies on Model-Driven Development with UML, and show the typical information found in articles of this topic.

The data collected during analysis of these articles was used to answer the following research question:

RQ: Is there support for the usefulness on Model-Driven Development with UML in empirical research, hereunder:

> **SRQ1:** What is the amount of empirical research on Model-Driven Development with UML in relevant journals and conference proceedings?
> **SRQ2:** What is the extent of the use of empirical experiments, case studies, surveys and experience reports in research on Model-Driven Development with UML?
> **SRQ3:** Which UML-diagrams have been evaluated and what are the benefits, if any?
> **SRQ4:** In which application domains has UML been evaluated?
> **SRQ5:** Is UML compared to other approaches?
> **SRQ6:** Is it possible to perform meta-analysis of parts of the research we'll find in this review?
> **SRQ7:** What does the authors of UML-studies claim to be important future work and what does this indicate about current research?

1.3 Contributions

The main contribution of this review is in presenting the state-of-the-art of Model-Driven Development (MDD) with UML.

The main conclusion of this study is that it does not exist sufficient empirical evidence to make conclusions regarding the usefulness of Model-Driven Development with UML. The few existing empirical studies of MDD with UML do mostly suggest that UML is useful, but they are too few and deal with too many aspects of UML-usage, to allow for definite conclusions regarding the usefulness of the technique.

Further, the main findings of this review are:

- 2,2 percent of the examined articles empirically evaluate Model-Driven Development with UML in industrial projects or in experiments with human subjects.
- Experiments are the most used empirical research method within studies of MDD with UML, and this study found four times as many experiments as case studies. In addition to case studies, it was also a number of experience reports and one structured questionnaire.
- The conference proceeding UML/MODELS, as the most important publication source for research of MDD with UML studied in this thesis, had primarily case studies and experience reports among the empirical studies. The most prestigious publication source that was examined in this selection, IEEE Transactions on Software Engineering, had only experiments.
- The most frequently evaluated UML-diagrams are Use Case Diagrams and Statechart Diagrams.
- Further, the existing studies deal with very many different aspects of UML usage. This makes it difficult to arrive at a conclusion regarding how to use UML and regards to utilitarian value and costs based on empirical studies. The empirical studies that involve UML usage have also often another primary focus on e.g. inspection of software artefacts.
- The overall results show that the use of UML has an impact on many aspects of software development, both in relation to comprehension, construction and correctness of a system and predictability in Software Engineering, and all the aspects have improvement potential when UML is used. However, such benefits are strongly dependent of the abilities and experience of developers and the application domain, which UML is applied.
- UML is not compared to any extent to other approaches. Only one article deliberatively compared UML to another approach. Three articles indirectly compared UML to other approaches.
- This study also looked at what the authors of the studies viewed as important future work. Most of them found it necessary to replicate the study, perhaps with another type of subjects or another application domain. Almost as many found it necessary to refine the UML-based technique under study. Only two of the studies argued that future studies should evaluate the cost-effectiveness of UML.

1.4 Structure of Thesis

Section 2 presents relevant background of Model-Driven Development with UML. Related work is presented in Section 3. An overview of empirical methods is presented in Section 4. The research method for this review is described in Section 5. Section 6 presents the findings and a discussion of the results of this review. Section 7 discusses the validity of this review. Finally, Section 8 concludes and presents directions for future work.

2. Model-Driven Development with UML

Model-Driven Development (MDD) aims to raise the level of abstraction for software development by the use of models as the key artefacts in software development, from system specification and analysis, to design and testing. Model Driven Development is increasingly gaining the attention of both industry and research communities. This thesis studies Model-Driven Development, with UML-models as the key artefacts in software development.

The following sections presents an introduction of the Unified Modelling Language and two related techniques.

2.1 The Unified Modelling Language

The Unified Modelling Language (UML) is a general-purpose visual modelling language that is used to specify, visualize, construct and document the artefacts of a software system [17]. UML captures decisions and understanding about systems that must be constructed, and is used to understand, design, browse, configure, maintain and control information about such systems.

UML is intended for use with all development methods, lifecycle stages, application domains and media [17]. The UML specification does not define a standard process but is intended to be useful with an iterative development process [17]. One such development process is the Rational Unified Process (RUP), which is developed hand-in-hand with the UML to guide the effective use of UML for modelling [14]. It describes which models you need, why you need them and how to construct them. RUP is also a Use Case driven approach, which means that the Use Cases defined for the system are the foundation for the rest of the development process.

The UML was adopted in 1997 as a standard by the OMG (Object-Management Group) and has continued to be refined in new versions, into today's UML 2.0. UML was developed in an effort to simplify and consolidate the large number of object-oriented development methods that had emerged and the modelling language is intended to unify past experience about modelling techniques and to incorporate current software best practises into a standard approach [17]. The Object-Management Group is also promoting a model-driven approach for software development through its Model Driven Architecture (MDA™) initiative and its supporting standards, such as UML, MOF and QVT [1]. With its rich palette and middleware independence, UML forms a foundation of MDA [2].

UML includes semantic concepts, notation and guidelines and has static, dynamic, environmental, and organizational parts [17]. It is intended to be supported by interactive visual modelling tools that have code generators and report writers. UML is built upon object-oriented concepts like classes and operation, however non-object oriented systems may also be modelled using UML.

A system is modelled as a collection of discrete objects that interact to perform work that ultimately benefits an outside user. The UML captures information about the static structure and dynamic behaviour of a system [17]:

- The static structure defines the kind of objects important for a system and to its implementation, as well as the relationships among objects to accomplish goals.
- The dynamic behaviour defines the history of objects to accomplish goals.

UML 2.0 defines thirteen types of diagrams, divided into three categories [2]:

Six diagram types represent static application structure; three represent general types of behaviour; and four represent different aspects of interactions:

> **Structure Diagrams**: Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.
>
> **Behaviour Diagrams:** Use Case Diagram; Activity Diagram, and State Machine Diagram.
>
> **Interaction Diagrams,** all derived from the more general Behaviour Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

UML provides several extension mechanisms to allow modellers to make some common extensions, to create tailored versions of UML, without having to modify the underlying modelling language. Extensions are organized into profiles. The extensibility mechanisms are stereotype, tagged values and constraints [17]:

- A stereotype is a new kind of model-element devised by the modeller and based on the existing kind of model element.
- A tagged value is a named piece of information attached to any model element.
- A constraint is a textual statement of a semantic relationship expressed in some formal language or in natural language. The UML includes the definition of a constraint language, the Object-constraint Language (OCL), which is convenient for expressing UML constraints.

A coherent set of stereotypes with their tag definition and constraints is modelled as a profile [17].

2.2 Related Techniques

UML was developed in an effort to simplify and consolidate the large number of object-oriented development methods that had emerged. Two of these methods are presented next. UML is partly based on these methods, among several other methods.

2.2.1 Specification and Description Language SDL

The Specification and Description Language (SDL) is an object-oriented, formal language defined by the International Telecommunications Union-Telecommunications Standardization Sector (ITU-T) as recommendation number Z.100. The key features of the language are summarized in [3].

Although SDL is widely used in the telecommunications field, it is also now being applied to a diverse number of other areas ranging over aircraft, train control, medical and packaging systems. The language is intended for the specification of complex, event-driven, real-time, and interactive applications involving many concurrent activities that communicate using discrete signals [4].

The basis for description of behaviour is communicating Extended State Machines that are represented by processes. For systems engineering SDL is usually used in combination with other languages and are comparative to a subset of UML.

2.2.2 Structured Analysis & Design

Structured Analysis and Design, abbreviated SA/SD has been the most popular and widely used analysis and design method since the 1970s. Although it is being superseded by object-oriented approaches, many of the notations, processes, and heuristics of this method have been adopted by later methods [5]. Also, SA/SD is still widely used.

Structured analysis and design is an approach that emphasizes analysis of data flows and processes rather than control flows or functional hierarchies [5].

The following diagrams of SA/SD are defined by [6]:

> **Data Flow Diagrams**: System analysts use process models (i.e. data flow diagrams, DFDs) to show information flow and processing in a system. The model usually starts with a context diagram showing the system bubble surrounded by the external environment identified by external entities. Data flows bring information to and from the system process. A process can explode to a child diagram that presents its details using data stores, data flows and sub processes. The diagram levelling process allows complex systems to be easily partitioned into a stack of simple diagrams with rigorous balancing of information between levels. Information structures are defined in an associated data dictionary.

Structure Charts: Structure charts show module structure and calling relationships. In a multi-threaded system, each task (thread of execution) is represented as a structure chart. Large structure charts are levelled into a stack of connected diagrams.

State Models: State models include diagrams and tables that show the significant states in a system, events that cause transitions between states and the actions that result.

Task Diagrams: Task diagrams show threads of execution and the real-time operating system services like queues, event flags and semaphores that connect them in a multi-tasking environment. Each task can be associated with its structure chart representation.

3. Related Work

There has been performed several surveys to determine the state of Software Engineering research as a whole with respect to topic, research approach, research method, reference discipline and level of analysis. This Section summarizes these efforts. These studies cover the entire field of Software Engineering, but are still of relevance to this thesis, due to the structure of the studies and the characteristics that have been measured. An overview of the related work can be found in Table 1.

Sections 3.1 to 3.5 give a description of the related work. A summary of the related work is provided in Section 3.6.

Table 1: Surveys of Empirical Studies in Software Engineering[1]

	Zelkowitz et al. [22]	Glass et al. [9]	Segal et al. [18]	Sjøberg et al. [19]	Holt [11]	This thesis
Purpose	Classifies empirical studies in SE and validates the taxonomy of empirical studies proposed by the authors	Surveys topics, research approaches, research methods, reference disciplines and level of analysis	Surveys topics, research approaches, methods, reference disciplines and level of analysis, units of analysis and authors.	Surveys topics, subjects, tasks, environments, and internal and external validity of controlled experiments in	Surveys the use of case studies in ESE.	Surveys the extent of empirical studies of Model-Driven Development with UML.
Scope	SE	SE	ESE	SE	ESE	ESE
Journals and proceedings	ICSE proc., IEEE Software, TSE	IEEE Software, IST, JSS, SP&E, TOSEM, TSE	EMSE	EASE, EMSE, ICSE, IEEE Computer, IEEE Software, ISESE, IST, JSME, JSS, METRICS, SP&E, TOSEM, TSE	EASE, EMSE, ICSE, IEEE Computer, IEEE Software, ISESE, IST, JSME, JSS, METRICS, SP&E, TOSEM, TSE	EMSE, UML/MODELS, TSE, RE
Sampling of papers	All papers in 1985, 1990 and 1995	Every fifth paper in the period 1995-1999	All papers between 1997 and 2003	All papers in the period 1993-2002	50 papers randomly selected among the papers scanned and analyzed by Sjøberg et al. [19]	All papers in the period 2000-2005
Number of investigated papers	612	369	119	5453 papers scanned, 103 papers analyzed in depth	427 papers scanned, 50 papers analyzed in depth	963 papers scanned, 21 papers analyzed in depth

[1] This table is an extended version of Table 1 in Sjøberg et al [19].

3.1 Research in Software Engineering: An Analysis of the Literature

Glass et al. [9] seek to give an objective description of the state of Software Engineering by examining 369 papers in six leading research journals in the Software Engineering field in the period 1995 to 1999. The papers were categorized according to topic, research approach, research method, reference discipline and units of analysis.

They conclude that SE research is diverse regarding topic, narrow regarding research approach and method, inwardly focused regarding reference discipline, and technically focused (as opposed to behaviourally focused) regarding level of analysis.

The spread of topics were broad. Most of the papers were placed in the category 'Systems/software concepts' (54.8 percent) where the subcategory 'methods/techniques' (18.2 percent) made the largest part.

As to research approach, over half of the papers were formulative (55.3 percent); a further 28% were descriptive and only 13,8% evaluative. Findings show that the most frequent used research methods are those concerning conceptual analysis and concept implementation. Laboratory experiments with human subject constituted only 3 percent, while the case study method constituted 2,2 percent.

Regarding reference disciplines, 98 percent of the papers did not have references to other fields. An interesting finding is that SE research is mostly about technical, computing focused issues, and rarely about behavioural concerns.

3.2 The Type of Evidence Produced by Empirical Software Engineers

Segal et al. [18] investigate the nature of the evidence published in the period 1997-2003 in the academic journal Empirical Software Engineering, drawing on the taxonomy developed by Glass et al. [9]. The 119 articles examined in [18] were classified according to topic, research approach, research methods, reference discipline and units of analysis.

Investigations of the following research questions were conducted; what is the prevalence of case and field studies of Software Engineering practice? Is there a wide variety in the types of evidence reported in the field of empirical Software Engineering?

The main findings of Segal et al [18] were the following:

- The research was somewhat narrow in topic with about half the papers focusing on measurement/metrics, review and inspection
- Researchers were almost as interested in formulating as in evaluating
- Hypothesis testing and laboratory experiments dominated evaluations
- That research was not very likely to focus on people and extremely unlikely to refer to other disciplines

Glass et al. [9] found that 13.8 percent of the papers featured evaluation, whereas Segal et al. [18] found that 53 percent of the papers in Empirical Software Engineering did the same.

3.3 A Survey of Controlled Experiments in Software Engineering

Sjøberg et al. [19] report on a survey that characterized quantitatively the controlled experiments in Software Engineering, published in nine journals and three conference proceedings (5453 articles) in the decade from 1993 to 2002. Only 113 (1,9 percent) of the 5453 articles reported controlled experiments. The study focuses on technology, subjects, tasks, type of application systems, and environments in which the experiments were conducted. Additionally, data on experiment replication, and internal and external validity were also collected and discussed.

The largest categories regarding topics are software lifecycle/engineering (49 percent) and Methods/Techniques (32 percent) caused by the large number of experiments on inspection techniques (36 percent) and object-oriented design techniques (eight percent).

It was found that 87 percent of the subjects were students whereas nine percent were professionals. Actually, almost 50 percent of all subjects in Software Engineering are students.

They identified tasks performed by the subject according to the following categories: plan (ten percent), create (20 percent), modify (16 percent), and analyze (54 percent). Duration of task was provided in some manner in almost 80 percent of the papers. However, specific duration data pr subject was only reported in 36 percent of the experiments.

In 75 percent of the experiments, the applications were constructed for the purpose of the experiment or were student projects. Commercial applications were used by 14 percent. Internal validity was reported in 63 percent and external validity in 69 percent of the experiments.

3.4 Experimental Validation in Software Engineering

Zelkowitz and Wallace [22] conducted a survey on experimental models for validating technology. By this study, they wanted firstly, to determine how well the computer science community is succeeding at validating its theories, and secondly, to determine how computer science compares to other scientific disciplines.

They developed a taxonomy for Software Engineering experimentation that describes the following twelve validation methods: static analysis, lessons learned, legacy data, literature search, field study, assertion, case study, project monitoring, simulation, dynamic analysis, synthetic and replicated. Additionally, a significantly amount of the papers were categorized as papers with no experimentation (papers describing a new technology that contained no experimental validations). The list was not meant to be

an ultimate list, rather as a good starting point for understanding Software Engineering experimentation. The study examined how these approaches have been used.

Of the 612 papers assessed, where 50 were judged to be “not applicable”, 562 papers were examined. These were published in IEEE Transactions on Software Engineering, IEEE Software and the proceedings from International Conference on Software Engineering from 1985, 1990 and 1995. Each paper was classified according to the data collection method used to validate the claims in the paper. They distinguished between data used as a demonstration of concepts and true attempts at validation of the results.

Zelkowitz and Wallace state among their quantitative findings that too many papers have no experimental validation (one third of the papers) at all. However, the percentage dropped from 1985 to 1995, which seems to indicate improvement. Among the papers that did have a form of validation, they claim that too many papers used an informal (assertion) form. Researchers use lessons learned and case studies in about ten percent of the studies, while the other techniques are used only sporadically. About five percent relied on the simulation method, while the remaining techniques were used in one to three percent of the papers. They also found that terminology is not used in a consistent manner.

The qualitative findings suggest that authors often fail to state their goals clearly or to point to the value that their method or tool adds to the experimentation process. Additionally, authors often fail to state how they validate their hypotheses and use terms very loosely.

3.5 A Systematic Review of Case Studies in Software Engineering

The work of Holt [11] is a systematic review of 50 randomly selected articles that report case studies. Holt [11] investigates the state of the art regarding the use of case studies in empirical Software Engineering. Secondly, important characteristics of case studies for researchers to give careful considerations when conducting case studies are identified.

Holt [11] has identified that research on technology that is to be adopted in an industrial setting must give evidence of relevance to the industry, and for this, case studies are important in that they give the opportunity to test technology in realistic surroundings with all the affecting factors. The data collected during analysis of these 50 articles, was used to address the following issues: the extent of case studies in empirical Software Engineering, the quality of reporting case studies, the specification of the case study research method, what researchers call a case study, the affiliation of authors, confusion regarding research methods, and the extent of the use of multiple case studies.

The main findings of Holt [11] are:

- Close to twelve percent of the 427 papers searched, use case study as the research method.
- There are great variances in the way of reporting case study results. The general impression is that information is not clearly reported.
- Researchers are not very likely to explicitly state what kind of research method that has been used.
- Case studies are mainly used for two purposes, namely evaluative and demonstrative purposes.
    - Typical characteristics for articles with an evaluative nature are rather high response rates for the six questions in the survey, the reporting of observations of use, and most likely the use of professionals as subjects.
    - Typical characteristics for articles with a demonstrative nature are relatively low response rates for the six questions in the survey, the reporting of technology outcome, and most likely the use of authors of the articles as subjects.
- The majority of the articles with authors affiliated in research communities appear to report technology data.
- The lack of observations of use may be reminiscent of the assertion method.
- The extent of multiple case studies is 22 percent.

Furthermore, Holt [11] suggest the following criteria for case studies in empirical Software Engineering:

> First of all, the author should specify that the research method used is the case study method. The focus in the case study should be use/evaluation of a software technology. Furthermore, the case study should test a technology in an industrial setting. Finally, the technology must be used by others than the researchers themselves (because of no manipulation), preferably by professionals.

Additionally, Holt [11] has identified a need for a specified definition of case studies standards for how to conduct case studies in empirical Software Engineering, and propose that use of guidelines would help researchers ensure the quality of the results.

3.6 Summary

As we can see there has been performed several surveys to determine the state of Software Engineering research as a whole with respect to topic, research approach, research method, reference discipline and level of analysis. The surveys express a general need for an increase in empirical validation in addition to a more structured way of reporting research.

There has however, to the authors' knowledge, not been undertaken any studies that thoroughly cover empirical studies of Model-Driven Development with UML. The classification scheme in e.g. Glass et al. [9] is for example not detailed enough to help us decide which parts of UML that is covered in the research.

The majority of the surveys I have referred to in this Section report on several types of research methods and the character of such studies in software engineering. Sjøberg et al. [19] and Holt [11] present an in-depth study of a specific research method, namely controlled experiments and case studies in Software Engineering.

A difference between this study and the studies I refer to is that I provide the state-of-the-art regarding the use of specific research methods and a specific topic, namely empirical experiments, case studies, surveys and experiences on Model-Driven Development with UML.

4. Empirical Research Methods

Empirical research could be defined as research based on the scientific paradigm of observation, reflection and experimentation as a vehicle for the advancement of knowledge. In this Section I concentrate on exploring empirical research methods and explain the importance for empirical methods in Software Engineering.

Wohlin et al. [20] state reasons for the importance of empirical methods in Software Engineering like this:

> "Software Engineering is not only about technical solutions. It is to a large extent also concerned with organizational issues, project management and human behaviour. For a discipline like Software Engineering, empirical methods are crucial, since they allow for incorporating human behaviour into the research approach taken."

Empirical methods provide an important scientific basis for Software Engineering. Empirical methods such as controlled experiments, case studies, surveys and experience reports are needed to help us evaluate and validate the research results. These methods are needed so that it is possible to scientifically state whether something is better than something else. The main motivation is that it is needed from an engineering perspective to allow for informed and well-grounded decision [20].

There are two main types of research paradigms having different approaches to empirical studies [20]:

- **Qualitative research** is concerned with studying objects in their natural setting. A qualitative researcher attempts to interpret a phenomenon based on explanations that people bring to them (Denzin and Lincoln references by [20]).

- **Quantitative research** is mainly concerned with quantifying a relationship or to compare two or more groups [Creswell references by [20]]. The aim is to identify a cause-effect relationship. The quantitative research is often conducted through setting up controlled experiments or collecting data through case studies. Quantitative investigations are appropriate when testing the effect of some manipulation or activity.

Quantitative strategies such as controlled experiments are appropriate when testing the effects of a treatment, while a qualitative study of beliefs and understandings are appropriate to find out why the results from a quantitative investigation are as they are [20].

The following sections describe common empirical research methods used in Software Engineering.

4.1 Experiments

In the scientific method, an experiment is a set of actions and observations, performed to verify or falsify a hypothesis or research a causal relationship between phenomena. They are often highly controlled and hence also occasionally referred to as controlled experiment [20]. Experiments are sometimes referred to as research-in-the-small [13] since they are concerned with a limited scope and most often are run in a laboratory setting.

Wohlin et al. [20] describes the operation of an experiment in the following way: "When experimenting, subjects are assigned to different treatments at random. The objective is to manipulate one or more variables and control all other variables at fixed levels. The effect of the manipulation is measured, and based on this a statistical analysis. In some cases it may be impossible to use true experimentation; we may have to use quasi experiments. The latter term is often used when it is impossible to perform random assignment of the subjects to the different treatments".

In an experiment the researcher has control over the study and how the participants carry out the tasks that they are assigned to. This can be compared to a typical case study, where the researcher is more of an observer [20]. The advantage of the experiment is, of course, that the study can be planned and designed to ensure high validity, although the drawback is that the scope of the study often gets smaller [20]. For example, it would be possible to view a complete software development project as a case study, but a typical experiment does not include all activities of such a project.

4.2 Case Studies

Case study research is sometimes referred to as research-in-the-typical [13]. It is described in this way due to that normally a case study is conducted studying a real project and hence the situation is "typical". Zelkowitz and Wallace [22] describe a case study to be an observational research method that is used for monitoring a project and collecting data over time without intervention by the researchers. This is in contrast to experiments, in which the researcher usually has control over various factors

Yin [21] defines a case study as follows:

1. A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.
2. The case study inquiry
   - Copes with the technically distinctive situation in which there will be many more variables of interest than data points, and as one result
   - Relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result
   - Benefits from the prior development of theoretical propositions to guide data collection and analysis.

Yin [21] states that case studies, like experiments, can be exploratory, descriptive or explanatory.

A case study is conducted to investigate a single entity or phenomenon within a specific time space. Data is collected for a specific purpose throughout the study. The researcher collects detailed information on, for example, one single project during a sustained period of time. Based on the data collection, statistical analyses can be carried out. The case study is normally aimed at tracking a specific attribute or establishing relationships between different attributes [20].

Within Software Engineering, case studies should not only be used to evaluate how or why certain phenomena occur, but also to evaluate the differences between, for example, two design methods. This means in other words, to determine "which is best" of the two methods [21]. Case studies are very suitable for industrial evaluation of Software Engineering methods and tools because they can avoid scale-up problems [20].

There are both pros and cons with case studies. Case studies are valuable because they incorporate qualities that an experiment cannot visualize, for example, scale, complexity, unpredictability, and dynamism [20]. Another advantage of case studies is that they are easier to plan but the disadvantages are that the results are difficult to generalize and harder to interpret, i.e. it is possible to show the effects in a typical situation, but it cannot be generalized to every situation [21].

When performing case studies it is necessary to minimize the effects of confounding factors. A confounding factor is a factor that makes it impossible to distinguish the effects from two factors from each other [20]. This is important since we do not have the same control over a case study as in an experiment.

The difference between case studies and experiments is that experiments sample over the variables that are being manipulated, while case studies sample from the variables representing the typical situation [20]. A case study is an observational study while the experiment is a controlled study, and further, the level of control is lower in a case study than in an experiment. Researchers are not completely in control of a case study situation. This is good, from one perspective, because unpredictable changes frequently tell them much about the problems being studied. The problem is that we cannot be sure about the effects due to confounding factors [20].

4.3 Experience Reports

An experience report/lessons learned is an historical method and is often produced after a large industrial project is completed, whether data is collected or not. The case study, in contrast, is an observational method that concerns the collection of data from projects as they evolve. A study of these documents often reveals qualitative aspects, which can be used to improve future developments [22]. If project personnel are still available, it is possible to interview them to obtain trends in looking at the effects of methods.

4.4 Surveys

In surveys, the primary means of gathering qualitative or quantitative data are interviews or questionnaires. A survey is by [13] referred to as research-in-the-large (and past) since it is possible to send a questionnaire to or interview a large number of people covering whatever target population we have. A survey is often an investigation performed in retrospect, when e.g. a tool or technique, has been in use for a while. Respondents belong to a representative sample from the population being studied. The results from the survey are then analyzed to derive descriptive and explanatory conclusions and then generalized to the population from which the sample was taken [20].

5. Methodology

Section 5.1 describes the research method I have used in the thesis. Section 5.2 present how the work of this thesis is distributed. Section 5.3 describes the planning of this review. Section 5.4 describes the selection of publication sources. Section 5.5 presents criteria and procedures for selecting articles, and Section 5.6 describes how the data was collected. Section 5.6 describes how the articles were analyzed. At last, Section 5.7 present interesting, unanswered questions.

5.1 Research Method - Systematic Review

As the purpose of this study is to investigate the extent of empirical research on model driven development with UML from the period 2000 to 2005, a systematic review was chosen as the research method for this thesis. The goal of this thesis is to present a review of current empirical evidence of Model-Driven Development with UML.

Kitchenham [12] propose a guideline for systematic reviews appropriate for Software Engineering researchers, including PhD students. These guidelines [12] have functioned as a guide for how to undertake this review.

Kitchenham [12] describes a systematic review as "a means of evaluating and interpreting all available research relevant to a particular research question, topic area or phenomenon of interest. Systematic reviews aim to present a fair evaluation of a research topic by using a trustworthy, rigorous, and auditable methodology." According to Kitchenham [12] the major advantage of systematic reviews is "that they provide information about the effects of some phenomenon across a wide range of settings and empirical methods". Another advantage, which is related to quantitative studies, is the possibility of combining data using meta-analysis techniques. This may increase the probability of identifying real effects that individual smaller studies are not able to detect.

A systematic review involves several discrete activities. Kitchenham [12] summarises the stages in a systematic review into three main phases:

1. Planning the review
2. Conducting the review
3. Reporting the review

The stages of a systematic review are explained next.

In the planning the review stage, the recognition of the need for a review is established. Moreover, a review protocol is developed. This protocol specifies the research question being addressed and the methods that will be used to perform the systematic review. The purpose of the protocol is to reduce the probability of researcher bias.

The components of a protocol include all the elements of the review plus some additional planning information [12]:

- Background. The rationale for the review.
- The research questions that the review is intended to answer.
- Search strategy (search terms and resources to be searched).
- Study selection criteria and procedures. It is usually helpful to pilot the selection criteria on a subset of primary studies.
- Study quality assessment checklists and procedures. Develop quality checklists to assess the individual studies.
- Data extraction strategy.
- Synthesis of the extracted data.
- Project timetable. Define the review plan.
- Protocol review. The protocol is a critical element of any systematic review. Researchers must agree a procedure for reviewing the protocol.

In the conducting systematic review phase, Kitchenham [12] emphasize the following activities:

1. The first activity is to identify the research, or, more precisely, the potentially relevant primary studies[2]. It is vital to determine and follow a search strategy for this identification process.
2. The second activity is the selection of the primary studies that are actually relevant.
3. When the relevance is decided upon, the third activity is to evaluate the quality of the primary studies.
4. The fourth activity is the data extraction. Here, it is important to have designed a data extraction form in order to accurately record information.
5. The fifth and last activity in the execution of the systematic review is the data synthesis. This activity includes gathering and summarising the results of the chosen primary studies.

Elements of the second, fourth and fifth activity should as far as possible be specified in the review protocol in the planning stage.

In the reporting stage of the systematic survey, the data are interpreted and presented. Kitchenham [12] emphasizes the importance of communicating the results of a systematic review effectively.

The various elements of the systematic review in this thesis are documented in sections 5.3 to 5.7.

[2] A primary study is the individual studies contributing to a systematic review [12].

5.2 Distribution of Work

The systematic review that is reported in this master thesis is partly a cooperation between two master students. The project group consisted of two master students and two supervisors. The planning and conducting of the review, selecting articles and data extraction, were undertaken by both students in the project. The cooperation lasted until data collection was finished (when all the wanted data was extracted from each of the included articles). The data analysis- and reporting stage were individual work.

When I in the next sections refer to "we", I refer to the work that was undertaken by both students. However, all the written parts of this thesis are individual work.

5.3 Planning the Review: Pilot Study

Before the systematic review could be conducted, we needed to get a basic understanding of the different empirical research methods, related studies and why the review was needed.

We were, before starting the examination of journals and conference proceedings, introduced to a sample of about ten articles from different publication sources. This collection was the choice of my supervisor and was meant as an ideal or inspiration-source for what type of studies that was to be interesting for this review. It was useful for us to get an idea of what to look for in subsequent work. We performed a pilot-study of these articles to decide upon research questions and what information we wanted to extract from the articles and decide upon criteria for inclusion. The result of this was a data extraction form with data fields to fill in data from the articles. We used these articles to identify and agree on an appropriate data extraction strategy and study selection criteria (inclusion criteria). The search strategy was also decided upon. To review the protocol, all the elements of this planning stage were agreed upon by the supervisors of this project.

The following sections describe the elements of the systematic that were agreed upon in the planning stage.

5.4 Selection of Journals and Conferences

We examined three journals and one conference proceeding in this review. The publication sources that were examined were chosen because they are central. It was decided that we would gather all articles of interest from these publication sources in the six- year period 2000-2005.

The four publication sources examined in this study have different focus and were selected for the following reasons:

- IEEE Transactions on Software Engineering (TSE) is the journal with most prestige within Software Engineering.

- Empirical Software Engineering (EMSE) is the leading journal for empirical studies within Software Engineering.

- UML/MODELS is the leading conference proceeding of Model-Driven Development (and it doesn't exist any corresponding journal).

- Requirements Engineering Journal (RE) is the leading journal on requirements engineering and we therefore expected that the journal would contain a few articles about especially Use Cases.

5.5 Study Selection Procedures and Inclusion Criteria

Once the appropriate decisions had been made regarding the duration of the study, the journals to be examined, and data extraction form and inclusion criteria to be used, we began examining the papers themselves. This section describes the search terms and study selection criteria for selecting the 21 articles that report on empirical studies of Model-Driven Development with UML.

First I will describe what type of studies that are included in this review. Next I will explain the procedures for deciding if an article presents an empirical study of Model-Driven Development with UML. At last is present the process of agreeing upon the inclusion criteria.

5.5.1 The Nature of Included Studies

In this thesis we are interested in including empirical studies of Model-Driven Development (MDD) with UML. Studies that are of interest in this review are studies that evaluate different properties, advantages and drawbacks of MDD with UML.

The types of empirical studies that are taken into account in this review are experiments, case studies, experience reports and surveys. We do not distinguish between randomized experiments and quasi experiments in this survey because both experimental designs are relevant to empirical Software Engineering experimentation. We are interested in studies that are of evaluative nature. Studies that are of no interest in this review are studies that are of demonstrative character. An article that proposes a new technique or approach and demonstrates the usage through a small example, a "case study," that only is performed to show the usability of the technology, is excluded from this review. This type of study is often called "proof of concept" and is often performed by the authors of the articles. This doesn't provide a realistic context and there could be a bias in favourite of the technology since the authors know their technology very well.

We found many articles that claimed to report a case study in the abstract, but then after reading the article, it appeared that the case study was of demonstrative character as described over, and it was excluded from the review. Often these articles proposed a new extension to UML or suggest a new approach without evaluate it properly. We are interested in studies that are performed in realistic contexts; this could be laboratory settings with human subjects or field studies/case studies with professionals or experience reports of UML in use.

5.5.2 Procedures for Selecting Articles

In order to identify and extract empirical research of Model-Driven Development with UML, we systematically read the titles and abstracts of 963 scientific articles published in the selected publication sources in order to identify and extract empirical studies done on Model-Driven Development with UML for the period 2000-2005.

The selection of articles was done in two iterations:

> First, we read through all the abstracts, and sometimes the introduction and conclusion of each paper to decide whether the article was interesting for our study based on keywords in its abstracts. These keywords are hereafter referred to as inclusion words. The inclusion criteria for the first iteration of selecting articles are a mix of subjective understanding of the paper when reading the abstract and use of inclusion-words.
>
> We gathered all abstracts in word-documents as we read them, one document for each journal or conference proceeding and sometimes one document for each year we examined. Each abstract was then marked with a colour code, green for included and red for excluded. Those abstracts that were marked green were then downloaded for further investigation. In this way we could easily cooperate, to decide what articles to include with all abstracts from a publication source gathered in one place. We could easily comment on the abstracts in the word-document and go back in time.
>
> Secondly, we examined each of the selected articles closer to decide whether they were to be included or excluded from the review. During this process we extracted data in order to answer the research questions from each article to fill in a data extraction form. The articles were also classified as experiment, case study or experience report. Those articles that were included in the first iteration but excluded in the next iteration were commented for why they were excluded.

The inclusion criteria for the first iteration were loosely stated. It is better to select too many articles in the first iteration than to miss articles that may be of interest as it is vital for the project to select all the existing articles of this topic.

The inclusion criteria varied over the sources we examined. Next I will describe the procedures for selecting articles from each of the publication sources.

The Journal Empirical Software Engineering

The Empirical Software Engineering Journal has a strong focus on empirical research and we could therefore expect to find empirical research. The focus of the first iteration when examining the abstracts in this journal was therefore to decide whether the articles reported on research of Model-Driven Development with UML, and not so much in deciding on the empirical value. We looked for words in the abstract that were in association with modelling and UML. These were words like e.g. UML, unified Modelling Language, modelling, Use Case, diagram, analysis and design and other words are that associated with e.g. UML-diagrams or other UML-constructs.

The Conference Proceeding UML/MODELS

The next publication source we examined was the conference proceeding UML/MODELS. The examination of this conference was much more troublesome than for the previous journal we examined as the focus of this examination was the opposite of the previous journal. Here we could expect to find UML-related research since the whole conference is dedicated to modelling and UML. We could, however, not expect anything about the empirical value of the articles. The main focus of this examination was therefore to decide the empirical value of each article. In this case we defined words that would make it easier for us to decide upon inclusion for the first iteration. The words are listed in Table 2.

If the paper looks interesting (regards UML or modelling), but doesn't include any of the inclusion-words, we may choose to take it into further investigation. If the abstract include both an inclusion word and an exclusion word, we include it based on the inclusion word.

Table 2: Inclusion- and Exclusion words

Inclusion words	Exclusion words
Empirical	Propose, proposed, proposal
Experiment	Present, presentation
Case study	Discuss, discussion
Experience report/lessons learned	Describe, description
Report on result/reports on/reports on the application	Introduce
	Address the problem/issue
Based on earlier research	Suggested, suggestion
Comparison/compare/comparative study	Identify
Take a systematic look	Explain
Investigate	Overviews/reviews
Analyze, analyse, analysis	Demonstrate/show how
Examines – Systematically evaluating	Illustrate/illustrated with
We study/Studies/studied	examples/demonstrate with examples/running
Assessment	example/demonstrate approach/exemplified
	A talk, argue, provide, explore, outline, characterize, define, consider

IEEE Transactions on Software Engineering and Requirements Engineering Journal

At last we examined the journals IEEE Transactions on Software Engineering and the Requirements Engineering Journal. In these two journals we could not expect anything about empirical value or content. We assumed that there wouldn't be that many UML-related articles so we selected in the first iteration all articles that had a hint of UML/modelling in it. The paper is included if the word «UML», «Unified Modelling language», «Use Case», «model-driven», "modelling" «PBR», «UBR» or another UML- associated word is present in the title or abstract of the paper. First we identified all the articles that studied UML and then we used the same inclusion words used in the previous section to decide upon the empirical value.

5.5.3 Deciding Upon the Inclusion Criteria

We used a bottom-up approach for choosing the criteria for inclusion. We had little experience of different research methods and how research papers are reported, so we partly defined the inclusion criteria while examining the papers. We had a good opinion about what kind of studies we wanted in our survey after presented to the pilot-study sample of articles but the complete set of criteria was decided after a period of reading articles.

As to come to a shared understanding of the criteria for inclusion we individually read the articles from both the pilot-study sample, the Empirical Software Engineering journal and the UML/MODELS conference individually and then came together to discuss our individual selection of articles.

It happened that we had selected different articles, and it was consequently useful to discuss the selected articles to come to a shared understanding of the inclusion criteria. If it was unclear from the title or abstract whether an empirical study of MDD with UML was described, we both read the entire article.

5.6 Data Extraction

During the pilot-study, when planning this systematic review, we identified what data we needed to extract from each article in order to answer the research questions. We identified both article-specific information and research method specific information that we needed to extract from each article. This resulted in a data extraction form with data-fields to fill in relevant information about each article. All the extracted data from each of the 21 included articles are presented in Appendix A.

First of all, we extracted article-specific data from each article. This includes the title of the article, authors, publication year, and publication source. Each article was also assigned to an article ID.

Further, we extracted data that was common for all research methods used in the articles. We extracted data if there existed an answer in the article. If not, the data field remained empty. The following data fields are common for all articles and research methods:

- **Type of study**, hereunder experiment, case study, experience report or structured questionnaire.
- **Intent:** The intension of the study.
- **Results/ Lessons learned:** The main results and lessons learned from the study.
- **Future work:** Directions for future work, what the researchers express about aims for future work, hereunder replication, refinement, comparing UML to other approaches, cost-effectiveness of UML etc.
- **Development phase:** In which development phase MDD with UML is applied, hereunder requirements, analysis and design, inspection etc.
- **Application domain:** Type of projects where MDD with UML are studied, hereunder-electronic commerce, telecommunication, embedded systems etc.
- **Participant/ project details:** Details regarding number of subjects, project details, education, experience etc.
- **Collection of data and analysis:** Description of how the study data were collected and analysed.

Furthermore, from experiments, case studies and the structured questionnaire we collected the same data, except from one extra data field that was collected exclusively from experiments, namely a field for which **Hypothesis** that were used in the experiment. These data fields were common for experiments, case studies and structured questionnaires:

- **Study design.**
- **Location:** Location of study, place or country.
- **Exp year:** Year of experimentation.
- **Duration:** Duration of study.
- **Case tools:** Case tools used in study.
- **Software artefacts:** Software artefacts used in study. Includes which UML diagrams that were studied.

For **Experience Reports**, we didn't include all of the data fields that were collected from experiments, case studies and structured questionnaire, but **Duration** of the experiences collected was extracted from these studies.

Some of the data fields became superfluous in this review. These data fields were supposed to be taken into account when answering whether a possible meta-analysis could be performed based on the included articles of this review (SRQ6). However, it appeared that the included articles were too heterogeneous to be subject for such procedures. The data fields that were supposed to be used when answering this question were; Participant details, Collection of Data and Analysis, Study design, Location, exp. year, Duration of study, Case tools and Hypothesis.

5.7 Analysis of the Articles

This section describes how the articles were analyzed in order to address each of the seven sub-research questions (see section 1.2).

The data extracted from the 21 articles selected from a selection of 963 articles are stored in MS Word documents, one document for each article. The word document consists of a template to fill in data that is collected from each article. It was our intention to make a database of the included articles, but due to the relatively small number of resulting articles and time-limitations, we didn't go through with this. It was a relatively easy task to withdraw data from each Word-document.

I used simple descriptive statistics on the collected data. For each article, I collected data to answer to each of these questions if an answer existed.

The total number of examined articles and the number of included articles were used to answer **SRQ1** about the amount of empirical research in Model-Driven Development with UML.

In order to address **SRQ2**, the extent of empirical research methods in MDD with UML, I classified each of the included articles according to research method used in study.

Furthermore, I classified the articles according to which UML-diagrams that were studied, what aspects of UML-usage that were evaluated and the results of the evaluation to answer **SRQ3.**

To answer **SRQ4**, I extracted data about which application domain UML is studied from each article.

To identify if UML is compared to other approaches, **SRQ5**, I collected data about the intention of the study.

SRQ6, regarding the possibilities for performing meta-analysis in this study, was addressed by examining the articles for homogeneity. This was done by examining hypothesis.

To identify **SRQ7**, what researchers aim for future work, I extracted data from each article on what they express about future work.

5.8 Other Interesting Questions

During analysis of the articles, many interesting questions were identified:

- Are UML diagrams useful in different contexts, like different application domains and development phases?
- How is a diagram used most beneficially? Are there benefits to be gained in applying UML-diagrams in different ways?
- Are there any benefits with combining diagrams to evaluate effect, maybe to improve e.g. comprehension or construction quality of UML?
- Are UML-diagrams beneficial when UML is compared to other approaches? Is there any benefits explored compared to not using modelling at all?
- What are the benefits when UML-diagrams are used in an extended way, e.g. in project management activities like estimation or in inspection of software artefacts?

These questions could, however, mostly not be answered based on the articles identified in this study.

6. Results

This Section presents and discusses the results of the review we conducted with the purpose of answering the research question presented in Section 1:

> ***RQ: Is there support for the usefulness of Model-Driven Development with UML in empirical research?***

The research question is further split into seven sub questions that will be paid attention to in Sections 6.1 to 6.7.

We investigated a total of 963 articles in three relevant journals and one conference proceeding, and ended up with 21 articles, a percentage of 2,2 percent, that fulfilled our criteria for inclusion. Each of the 21 articles have been assigned to an ID, in the format A#, which will be used to refer to the analyzed articles in the following sections. The data extracted from these articles can be found in Appendix A.

6.1 The Amount of Empirical Research on Model-Driven Development with UML

In this section I investigate the extent of empirical research on Model-Driven Development (MDD) with UML in the examined journals and conference proceeding. Table 3 presents a summary of the extent of included articles in the four publication sources that was examined.

Table 3: The Extent of Included Articles and Research Methods in each of the examined Publication Sources.

Journals and conference proceedings	Total No of articles 2000-2005	No of articles included in the review	Distribution of included articles on research methodology
Empirical Software Engineering (EMSE)	139	**7**	6 Experiments 1 Structured questionnaire
UML/MODELS	228	**8**	5 Experience reports 1 Case study 2 Experiments
IEEE Transactions on Software Engineering (TSE)	462	**2**	2 Experiments
Requirements Engineering (RE)	134	**4**	2 Experiments 2 Case studies
TOTAL	**963**	**21**	12 Experiments 5 Experience reports 3 Case studies 1 Structured questionnaire

The journal IEEE Transactions on Software Engineering (TSE) contained the largest number of articles, with approximately half of the total number of examined articles. Only 2 out of 462 articles in TSE were included in this review. This is a share of 0,43 percent. The TSE Journal was the source with the lowest share of empirical research of Model-Driven Development with UML among the four publication sources that were investigated.

In the Empirical Software Engineering Journal, 7 out of 139 articles were included in the review. This is a share of five percent of the articles, and was the publication source with the highest share of empirical research of Model-Driven Development with UML.

The conference proceeding UML/MODELS is the leading conference proceeding of Model-Driven Development (and it doesn't exist any corresponding journal). UML/MODELS contained few articles with empirical value. Only 3,5 percent, 8 out of 228 articles in UML/MODELS reported on empirical studies of MDD with UML.

The Requirements Engineering Journal contained a total of 134 articles, and the included articles from this journal were counted to four. This is a share of almost three percent.

The results do not show a big difference across the four publication sources in terms of the amount of empirical research of Model-Driven Development with UML. Three of the publication sources have a share of between three and five percent of empirical studies of interest for this review. The fourth publication source, namely the TSE Journal, separates from the three other sources with a significantly smaller share of empirical studies of Model-Driven Development with UML. The TSE Journal is also the publication source examined in this review that has the widest perspective, both in terms of number of articles and scope. The other three publication sources are more narrowed towards specific domains, like empirical Software Engineering, Model-Driven Development and Requirements Engineering.

The fact that empirical studies of Model-Driven Development is represented so scarcely in the TSE Journal indicates that empirical studies of Model-Driven Development play a minimal role in the Software Engineering community, as the TSE Journal is a publication with much prestige within the Software Engineering community.

UML/MODELS is the leading conference proceeding of Model-Driven Development. For that reason we expected to find a higher share of empirical studies of Model-Driven Development in the UML/MODELS conference compared to the other three publication sources, which cover a wider part of the field of Software Engineering. It seems that empirical methods are not central among those who are interested in UML, and that UML-research is focused on formulating and proposing new methods and extensions to UML and not so much on evaluating existing approaches.

From the Requirements Engineering Journal we expected to find studies of Use Case modelling, but the journal has neither focus on empirical research nor UML.

It appears that Model-Driven Development with UML is not a central topic among researchers who have focus on empirical studies. Correspondingly, empirical methods are not in widespread use among those who are interested in UML. Further, UML-research is focused on formulating and proposing new methods and extensions to UML and not so much on evaluating existing approaches.

Trend over Years

I can see no trend in any directions for the years 2000 to 2004. In the year of 2005, however, I found twice as many articles as the preceding years. Almost half of the included articles (43 percent) were found in the year of 2005. Table 4 presents the distribution of included articles over the years. It seems that the empirical research of Model-Driven Development with UML is becoming more widespread in these days, but it is too soon to say whether this is a coincidence or if it is to become more focus on this type of research.

Table 4: Trend over Years

Year	No of Included Articles
2000	4
2001	2
2002	3
2003	1
2004	2
2005	9

6.2 The Extent of Experiments, Case Studies and Experience Reports

In this section I investigate the extent of empirical experiments, case studies and experience reports in Model-Driven Development research.

The research method that is most represented in the included studies are experiments. Twelve articles, which is over half of the included articles, use experiments as research method. The other half part of the articles is distributed between five experience reports, three case studies and one structured questionnaire.

I found an uneven distribution of experiments, case studies and experience reports across the examined journals and the conference proceeding. The distribution of different research methodologies across publications can be seen in Table 3.

The journals Empirical Software Engineering (EMSE) and IEEE Transactions on software Engineering (TSE) contained a higher percentage of experiments than the other two publication sources. EMSE and TSE contained eight of the twelve experiments included in the review, and had no share of case studies or experience reports. EMSE contained the majority of experiments that were included in the study. Six out of twelve articles that reported experiments were found in the EMSE journal. EMSE didn't have case studies or experience reports, but contained one study based

on the use of a structured questionnaire. The other three publication sources contained two experiments each.

UML/MODELS is focused more on experiences and case studies. UML/MODELS contained all the experience reports included in the review. Case studies are distributed between the Requirements Engineering Journal (2) and UML/MODELS (1).

Table 5: Research Methods in Included Articles.

Research Methodology	Article IDs	No of Articles	%
Experiment	**A5, A7, A9, A10, A12, A13, A14, A15, A16, A17, A19, A20**	12	57,1
Experience Report	**A1, A3, A4, A6, A8,**	5	23,8
Case Study	**A2, A18, A21**	3	14,3
Structured Questionnaire	**A11**	1	4,8
TOTAL		**21**	**100**

6.2.1 The Extent of Experiments

The amount of included experiments in this review is 1,25 percent of the total number of examined articles. The extent of experiments varies from 0,4 percent to 4,3 percent across the four examined publication sources.

The surveys summarized in Table 1 also report extent of experiments in various studies:

In Glass et al. [9], the authors classify 3 percent of the articles as laboratory experiments using human subjects and less than 1 percent as field experiment. According to the survey by Zelkowitz and Wallace [22], laboratory experiments as controlled methods are reported in 2,1 percent of the articles. Sjøberg et al. [19] find a lower percentage of articles (1,9 percent) that report controlled experiments.

Like the results in Sjøberg et al [19], this review report a higher proportion of controlled experiments for the EMSE journal than for the other examined sources. The percentage for controlled experiments in EMSE is 4,3 percent in this review. This is not surprising as the focus of the EMSE journal is empirical Software Engineering. Since the EMSE journal stand out to report a higher percentage of experiments, it is interesting to compare the results from this review with other related studies. According to the survey by Sjøberg et al. [19], the percentage of controlled experiments in EMSE is 17,7 percent. Segal et al. [18] report that laboratory experiments with human subjects are reported in 29 percent of the articles in EMSE. The differences between Segal et al. and Sjøberg et al. might be the narrower study type definition of Sjøberg et al. Sjøberg et al. [19] have probably used a narrower type definition of experiments, so the results of this study are probably most comparable with the Segal et al. [18] study. Thus, this means that experiments of MDD with UML is represented in 4,3% of the articles in EMSE, compared to all experiments in EMSE which is 29%.

6.2.2 The Extent of Case Studies

The amount of included case studies in this review/Model-Driven Development is **0,3 percent**.

Other studies have examined the use of case study as research method in the field of Software Engineering. .

- Segal et al. [18] found that **13 percent** of the papers examined in the Empirical Software Engineering journal, used case study as the research method.
- Holt [11] found that close to **12 percent** reported on case studies.
- Zelkowitz and Wallace [22] found that **10,3 percent** of the papers relied on the case study method.
- Glass et al. [9] found that only **2.2 percent** of the papers were case studies.

The share of case studies and experiments for MDD in this study seem to agree proportionate with the share of the total distribution of empirical studies in Software Engineering, but it appears to be a somewhat larger share of experiments in the sub-field MDD with UML compared to the whole field of Software Engineering.

6.2.3 The Extent of Experience Reports

The amount of included experience reports in this review is **0,5 percent**. It is difficult to state something general about this number, since the share of experience reports are generally not so closely investigated.

6.3 Evaluated UML-Diagrams and Reported Benefits

In this research question I am interested in investigating the extent to which the different UML diagrams are evaluated. I am also interested in investigating what aspects of UML-diagram usage that is evaluated and what the possible benefits (and drawbacks) with different diagrams are.

Over the years, UML has naturally developed in certain directions from the first version in 1997 to today's version, UML 2.0. The UML-diagrams have consequently also changed over time. Names of UML-diagrams have slightly differed with different versions of UML. When I next speak of different UML-diagrams, I have not taken into account different versions of UML, but I simply refer to the diagram-names used in each article.

UML-diagrams are evaluated in 11 out of 21 articles, whereby one is a case study and ten are experiments. The other ten articles evaluate UML as a whole or evaluate extension mechanisms of UML such as stereotypes, meta-models/UML-profiles or tools etc.

I found that seven UML-diagrams are evaluated in different ways in the 21 included articles of this review. As a comparative reference, but not an answer, OMG [2] defines 13 UML-diagrams in their UML 2.0 version. The distributions of UML-diagrams that are evaluated in the included articles are presented in Table 6.

Table 6: UML-Diagrams Evaluated

Diagrams Evaluated	Article IDs	No of articles
Class Diagram	A14, A15, A17	3
Deployment Diagram	A15	1
Use Case Diagram	A2, A9, A10, A13, A14, A16, A17	7
Statechart Diagram	A5, A12, A15, A17	4
Sequence Diagram	A12, A15, A17	3
Collaboration Diagram	A12	1

- The most evaluated UML-diagram in this review is the **Use Case Diagram**, where 7 out of the 21 included articles have evaluated Use Case Diagrams in one way or another.
- The second most evaluated UML-diagram is the **Statechart Diagram**, which is evaluated in four articles.
- The **Class Diagram and Sequence Diagram** is both the third most evaluated diagram in this review with three articles each.

Many articles evaluate more than one diagram-type in their study. The 11 articles that evaluate UML-diagrams are represented 18 times in Table 6. This means that each article evaluates almost an average of 2 diagrams each.

A list of articles that study different diagrams, and what aspects that are evaluated are presented in Table 7.

Table 7: Articles that evaluate UML-Diagrams.

Article ID	Intent	Diagrams studied	Aspects evaluated
A2 Case study	Evaluate a method for effort estimation based on Use Cases	Use Cases.	Accuracy
A5 Experiment	Evaluate the effect of composite states in Statechart diagrams.	Statecharts.	Understandability and efficiency
A9 Experiment	Evaluate the use of a set of Use Case authoring guidelines	Use Cases.	Construction completeness and structure.
A10 Experiment	Evaluate an inspection technique that is based on Use Cases.	Use Cases.	Defect detection efficiency.
A12 Experiment	Evaluate three different notations for representing the dynamic behaviour in UML	Sequences, Collaborations, and Statecharts.	Semantic comprehension. Time and score.
A13 Experiment	Evaluate an inspection technique that is based on	Use Cases.	Defect detection efficiency.

	Use Cases.		
A14 Experiment	Evaluate two alternative ways of applying a Use Case model in a design process.	Use Cases and Classes.	Completeness, structure and time.
A15 Experiment	Compare an approach made by authors' to UML.	Deployments, Statecharts, Classes, Sequences.	Comprehension and construction quality
A16 Experiment	Evaluate an inspection technique that is based on Use Cases compared to another approach.	Use Cases.	Defect detection efficiency.
A17 Experiment	Evaluate the impact of OCL in UML.	Use Cases, Sequences, Statecharts, and Classes.	Comprehension, Maintenance, and Defect Detection effectiveness
A19 Experiment	Evaluate the possible synergies and relationships between diagrams in the context of requirements analysis.	Use Cases and Classes.	Complementary effect, Informational roles and values. Completeness of diagrams and perceived ease of use.

6.3.1 Different Ways UML-Diagrams are Evaluated

I have identified three main categories for how UML-diagrams are evaluated with regards to the usage area of different diagrams. These categories are explained next, and the different articles in each category are listed.

The first category is articles that evaluate diagrams "as they are" and in their usual area of use:

- Compare different ways of applying UML-diagrams to investigate which way that leads to a better result in other UML-diagrams [A14].
- Compare UML to other self-made technique, with regards to comprehension and construction quality in UML-diagrams of web application models [A15].
- Evaluate the semantic comprehension of diagrams [A12].
- Evaluate the synergies and relationships between diagrams in the context of requirements analysis [A19].

The second category is articles that evaluate UML-diagrams when different guidelines, languages, methods, techniques, formality, or UML-extensions etc. are added or applied to the diagrams to evaluate effect:

- Evaluate the understandability of Statechart Diagrams when composite states are applied [A5].
- Evaluate construction-, completeness-, and structure quality of Use Case descriptions when a specific Use Case authoring guideline is applied [A9].
- Evaluate whether OCL has an impact on defect detection in UML models, comprehension of UML models and maintenance of UML documents [A17].

The third category consist of articles that evaluate extended use of UML-diagrams e.g. where diagrams are applied to process activities or development phases like estimation, inspection or other activities that are beyond the ordinary usage area:

- Evaluation of how well Use Cases (as input to the Use Case Point method) apply to the estimation process [A2].
- Evaluation of how well Use Cases apply to inspection of software artefacts [A10, A13, A16].

6.3.2 Aspects Evaluated

The included articles evaluate different aspects of UML-diagram usage. Some articles evaluate comprehension aspects of using UML-diagrams, while others evaluate construction aspects.

Except from four articles, all the included articles evaluate comprehension or/and construction aspects of UML-diagrams in addition to other aspects such as maintenance, defect detection and synergies between diagrams. I found that these two aspects, comprehension and construction, were the two main aspects that are evaluated.

The four articles that don't evaluate comprehension and construction aspects of UML-diagrams evaluate extended use of UML where UML-diagrams are used as a basis for methods that is used in wider development such as the estimation process and inspection of software artefacts. The aspects studied in these four articles are defect detection, time and accuracy aspects.

When it comes to comprehension and construction quality of UML diagrams, I can see a few tendencies for what type of diagrams that are likely to be evaluated related to certain aspects. Table 8 gives an overview of what aspects that are studied in which article.

When I look at the **comprehension aspect** of evaluated UML-diagrams, I see that the most evaluated diagram is the Statechart Diagram. Sequence Diagrams and Class diagrams are evaluated second most frequently. When it comes to **construction aspects** of UML-diagrams, however, two other diagrams stand out as the two most evaluated diagrams, namely Use Case- and Class Diagrams.

The **Statechart Diagram** is the most evaluated UML-diagram when we look at comprehension aspects. All the four articles that reported on evaluation of comprehension aspects had Statechart diagrams as part of the evaluation. But the Statechart Diagram was only evaluated exclusively in one article. The three other articles evaluated Statechart Diagrams along with other diagrams. The Statechart Diagram is also evaluated against construction, defect detection, maintenance and time aspects. It is not evaluated with regards to accuracy of method, ease of use and synergic values.

From Table 8 it appears that the **Use Case Diagram** is the diagram that is evaluated with the widest perspective. The Use Case Diagram is evaluated with respect to all the aspects of UML-diagram usage that I have registered. The Use Case Diagram is also

the only diagram that is used in extended ways in wider development activities such as estimation of effort and inspection of software artefacts. In estimation, Use Cases are a basis for the Use Case Point method that is used to estimate effort. Use Cases are also used in inspection of software artefacts in methods such as Usage-based reading and Perspective-based reading. Use Cases are here used to guide the inspection of software artefacts.

Table 8: What Aspects that are Studied in which Article

Aspects studied	Article IDs	No of articles a diagram is studied
Comprehension	A5, A12, A15, A17, A19	Statechart Diagram – 4 Sequence Diagram – 3 Class Diagram – 3 Use Case Diagram – 2 Deployment Diagram – 1 Collaboration Diagram – 1
Construction	A9, A14, A15	Use Case Diagram – 2 Class Diagram – 2 Statechart Diagram –1 Sequence Diagram – 1 Deployment diagram – 1
Defect detection	A10, A13, A16, A17	Use Case (PBR/ UBR) – 3 Use Case Diagram – 1 Sequence Diagram – 1 Statechart Diagram – 1 Class Diagram – 1
Maintenance	A17	Use Case Diagram Sequence Diagrams Statechart Diagram Class Diagram
Accuracy of method	A2	Use Case Point Method – 1
Time aspects	A10, A12, A13, A14, A16	Use Case (UBR/PBR) – 3 Use Case diagram – 1 Sequence Diagram- 1 Collaboration Diagram – 1 Statechart Diagram – 1 Class Diagram – 1
Synergies	A19	Use Case Diagram – 1 Class Diagram – 1
Ease of use	A19	Use Case Diagram – 1 Class Diagram – 1

6.3.3 Benefits of Using UML-Diagrams

In this section I will answer the research question whether UML-diagrams yield any benefits under development of new and existing software. More detailed questions that we would like to find answers for in empirical studies of UML are:

1. Do different UML-diagrams yield benefits in different ways, and what are the possible drawbacks with different diagrams?
    a. Are there different benefits to be gained when applying UML-diagrams in different ways?
2. Which diagrams gain benefits in terms of comprehension and construction quality?
    a. Are some diagrams easier to comprehend or to construct than others and are certain diagrams more completely constructed than others?
    b. Are some diagrams easier to learn for amateurs?
3. What are the benefits when UML-diagrams are used in an extended way, e.g. in project management activities like estimation or in inspection of software artefacts?

First of all I will summarize the results of evaluated aspects like comprehension and construction quality of UML-diagrams. Next I will summarize the result of defect detection aspects and accuracy of method aspects.

Results from Evaluation of Comprehension Aspects

When it comes to comprehension aspects of modelling with UML- diagrams, I have summarized all the results:

- The use of Composite states improves the understandability of Statechart Diagrams, so long as the subjects have some previous experience in using them [A5].

- The comprehension of the dynamic modelling in object-oriented designs depends on the diagram type and on the complexity of the document. The design is more comprehensible, when the dynamic behaviour is modelled in a Sequence Diagram. When using a Collaboration Diagram, the design turns out to be less comprehensible as the application domain, and consequently, the document is more complex [A12].

- For structure comprehension, when comparing another self-made technique (OPM) to UML, the results were better for OPM in one system; but better for UML in the other [A15].

- Using OCL can improve the ability to understand a system modelled with UML, but such benefits are strongly dependent on ability, experience, and training [A17].

- Use Case Diagrams are more completely interpreted than Class Diagrams, and the presence or absence of one diagram (out of Use Cases and Class Diagrams) when interpreting the other diagram has no effect on the outcome of the interpretation [A19].

The overall results are, hence, that different ways of applying UML diagrams and augmenting them with Composite states or OCL appears to have an impact on the comprehension of software systems, but the results are very much dependent on the application domain for which UML is used and the qualifications of the developers applying the technique. The results also show that diagrams do not support the understanding of other diagrams.

Results from Evaluation of Construction Aspects

The results from evaluating of construction aspects are summarized next:

- Use Case authoring guidelines do not necessarily improve the Use Case descriptions [A9].

- Different ways of applying a Use Case model in an object-oriented design process have an impact on the quality of the resulting Class Diagrams [A14]: Using Use Cases in the validation of Class Diagrams resulted in Class Diagrams that implemented more of the requirements, while deriving Class Diagrams from Use Cases resulted in Class Diagrams with a better structure.

- The results suggest that the technique made by the authors, OPM, is better than UML in modelling the dynamics aspect of the Web applications [A15].

The existing results give some indications of how different ways of applying UML in the construction process may yield differences in the quality of the final product, but overall the results show that the construction aspects of UML have been the subjects of very little evaluation.

Results from Correctness Aspects

Four articles evaluate defect detection aspects of UML-diagrams.

Three articles evaluate how well Use Cases apply to the inspection of software artefacts [A10/A13/A16]:

One article investigates if different perspectives taken by inspectors lead to the detection of different defects and if one perspective is superior to the other [A10]. The results show that the perspective that uses Use Cases didn't succeed more than the other two perspectives. The two other evaluate an inspection technique based on the expected usage of the system, Usage-based reading [A13/A16]. The results of these studies show that Usage-based reading is efficient and effective in detecting the most critical faults from a user's point of view, and it is also most efficient to use pre-developed Use Cases for usage-based reading [A13].

One article [A17] evaluates the impact of Object-constraint language, OCL, on the effectiveness when detecting defects in UML-diagrams. The results of this study show that OCL has the potential to significantly improve an engineers' ability to inspect a system modelled with UML, but also such benefits are strongly dependent on the ability, experience, and training of software engineers [A17]

The overall results show that applying OCL in the construction of UML-diagrams may improve such models, and further that applying usage-based reading with Use Cases as input may be a way of improving defect detection in code inspections. However, the results also show that the individual abilities and motivation of those using the techniques were more important than the actual technique used.

Use Case Diagrams in Predictions

One article has studied the accuracy of an estimation method based on Use Cases, the Use Case Point method, in estimating software development effort [A2]. The results of this study indicate that the Use Cases can be useful in improving estimation accuracy.

6.4 UML Studied in Different Application Domains

This section presents an overview of in which application domains that UML have been evaluated.

Nine of the articles report on studies of UML in different application domains. These nine articles and the application domains studied are listed in Table 9.

Table 9: Application Domains Studied.

Article ID#	Application Domains Studied
Case Studies	
A2	Electronic commerce and call-centres, in particular within banking and finance.
A18	Electronic commerce.
A21	Web- application. A Web-enabled database with end user and call centre operator interfaces.
Experience Reports	
A1	A global customer service system and an Embedded control system for flow meters for a large, globally distributed shipping company.
A3	Thales - a wide variety of systems.
A4	Control system for Industrial plant applications.
A6	Telecommunication.
A8	A Commercial Customer Relationship Management application /a service-oriented, multi-tier application.
Structured Questionnaire	
A11	Telecommunication.

It appears that a few application domains like telecommunication; electronic commerce and control systems are studied more frequently than other application domains in the included articles of this review.

When it comes to experiments, one out of twelve experiments study UML for specific application domains. This experiment use extensions of UML for Web-applications [A15]. One of the other articles [A7] studies quality of UML-stereotypes, and presents an experiment that is independent of application domain.

The rest of the experiments, ten articles, use "toy systems" as experimental tasks in their experiments. Six out of ten articles use either ATM systems or Library systems as experimental tasks in their experiments.

The most frequently used "toy systems" are ATM [A5, A10, A19] and Taxi systems [A13, A16, A17] with three articles each using these two as experimental tasks. Library systems are also popular with two articles using them [A12, A14].

Other "toy systems" that are used as experimental tasks are:

- A phone call [A5].
- Interaction between a supermarket checkout operator and the checkout machine [A9].
- PG (Parking garage) [A10].
- A Simple Cellular Telephone [A12].
- A Digital Dictaphone [A12]
- A Video Store (VS) system [A17].
- A music club [A19].
- Socio-technical systems for the Military domain [A20].

6.5 UML Compared to Other Approaches

The results show that UML is not empirically compared to other approaches to any great extent. Only one article has deliberately tried to compare UML to another approach. This article experimented with comprehension and construction of web application models, and compared a technique made by the authors, OPM, to UML [A15].

Of the 21 included articles, it is three articles that indirectly compare UML to other approaches. These three articles report on studies with Usage-based reading and Perspective based reading and has evaluated how well Use Cases apply to inspection of software artefacts. Usage-based reading and Perspective-based reading are reading techniques that utilize Use Cases during fault searching. The cornerstones of UBR are Use Cases and prioritization. Use Cases are utilized to guide reviewers through a software document during inspection.

Two of these articles report on studies that perform a comparison between Usage-based reading and checklist-based reading. Checklist-based reading is the traditional reading technique that provides a list of issues and questions, capturing the knowledge of previous inspections, helping the reviewers to focus their reading.

The third article investigates whether different perspectives in Perspective-based reading (PBR) detect different defects and if one is superior to the other. One of the perspectives in the PBR reading technique is the user-perspective, which applies Use

Case modelling. Here, the user perspective is compared to equivalence partitioning for the tester perspective and structured analysis for the design perspective.

The Use Cases are prioritized in an order of importance from users' requirements on the system developed. Hence, reviewers using UBR focus on the important parts first, leading to the important faults are found.

We had expected to find more articles that compare UML to other approaches. Examples of such approaches are the Specification and Description Language SDL [3, 4] and Structured Analysis & Design [5, 6]. These two modelling languages are presented in section 2.2.

6.6 Meta-Analysis

One question I want to investigate is whether there is a basis for drawing conclusions from research done of Model-Driven Development with UML. In this section I investigate the possibilities for performing procedures for meta-analysis in this thesis.

Meta-analysis is a statistical procedure for combining data from multiple studies. Miller [15] states reasons for performing meta-analytical procedures like this:

> "Deriving reliable empirical results from a single experiment is an unlikely event. Hence to progress multiple experiments must be undertaken per hypothesis and the subsequent results effectively combined to produce a single reliable conclusion."

Decisions about the utility of an intervention or the validity of a hypothesis cannot be based on the results of a single study, because results typically vary from one study to the next [7].

It is interesting to investigate whether it is possible to perform meta-analysis of the research in the field of Model-Driven Development. The main conclusion I have arrived at is that the included articles of this review are too different and too few to be object for meta-analysis. Among the 21 included articles, there are a great variety of UML related topics studied. It is a necessary demand for certain homogeneity across studies, when meta-analysis is taken into account, and the included articles of this review are too heterogeneous.

In their illustration of the use of meta-analysis in Software Engineering, Pickard et al. [16] conclude that:

> "Meta-analysis is appropriate for homogeneous studies when raw data or quantitative summary information, e.g., correlation coefficient, are available. It can also be used for heterogeneous studies where the cause of the heterogeneity is due to well-understood partitions in the subject population."

When the treatment effect (or effect size) is consistent from one study to the next, meta-analysis can be used to identify this common effect [7]. When the effect varies from one study to the next, meta-analysis may be used to identify the reason for the variation.

The only subject that is evaluated a few times in the included articles are reading techniques, based on use-cases, that is used in inspection of software artefacts. These articles are however not homogenous enough to be object for meta-analysis in this thesis.

Meta-analytic methods allow us to summarize the outcomes of previous research, and form empirically derived expectations for future research focus [10].

Next I will present some procedures for meta-analysis.

Miller [15] describes the starting procedures for meta-analysis as the following stages:

1. A traditional meta-analysis starts with an exhaustive search of the literature to find all the articles describing empirical evaluations of the concept under investigation.

2. Subsequently the researcher must analyse these reports for their quality and the `weight' of their experimental results, but initially what is important is that all the potentially relevant articles are found.

3. From these articles, the researcher should only identify the relevant variables required to evaluate their meta-analytical hypothesis. The unit of analysis in a meta-analysis should be the impact of variable X on variable Y.

4. Once the researcher has completed these processes, they are ready to start the meta-analysis process.

I have performed stage one of these procedures. As the procedures described by [15] indicate; a natural step following a systematic review is meta-analysis. Meta-analysis would also be a natural step following this systematic review, but the articles of this study are too heterogeneous to be object for meta-analysis.

Glass et al. [9] investigated the research methods used by Software Engineering researchers in the period 1995 to 1999. They found that meta-analysis was represented in zero percent of the research methods used. This indicates that empirical Software Engineering, MDD with UML included, has little focus on combining and summarizing empirical results.

6.7 Directions for Future Work

In this section I investigate what the authors' of the included articles aim for future research and what they advise others to focus on. I want to find out where the focus is placed when the experts consider future work and what this indicates about current work and what directions MDD research with UML is taking.

What the researches express about important future work indicate how far the research has come. Often the few sentences in the end of each paper say a lot about the quality of the work done, what the authors didn't have time for in their current study, weaknesses with their study and what the author consider being important unexplored aspects of UML usage.

To answer these issues I have divided this sub-research question into eight questions that are answered in the following sections.

6.7.1 The Amount of Articles that Present Aims for Future work

All the 21 included articles present some kind of directions for future work. Most of the authors aim to conduct further studies in the future to validate the results of their work and contribute to the body of knowledge of Model-Driven Development with UML. It varies, however, how detailed the aims for future work is reported. Table 10 presents an overview of what different articles aim for future research. As we can see, the authors' focus mainly on replicating their study and refining the method under consideration.

Table 10: Articles and Aims for Future Work

Goal for Future Studies	Article IDs	No of Articles	%
Replicate in other contexts	A1, A2, A5, A7, A10, A11, A12, A13, A14, A15, A16, A19	12	57,1
Replicate, but not specified how	A9, A17, A18, A21	4	19
Refine method under study	A1, A2, A3, A5, A7, A8, A9, A18, A20, A21.	10	47,6
Replication and refinement	A1, A2, A5, A7, A9, A18, A21	7	33,3
Compare UML to other approaches	A2, A4, A15, A16	4	19
Combine models/methods	A2, A16, A21	3	14,3
Wider perspective	A2, A21	2	9,5
Cost-effectiveness	A2, A17	2	9,5

6.7.2 Replication of Study in Different Contexts

In this section I identify how many authors' that aim to replicate their study in other contexts (different domains, different subjects, different development phase, different research methodology etc.) to contribute to the body of evidence on Model-Driven Development with UML. Is this work in progress? Table 11 presents an overview of how the authors intend to differentiate their future replications.

A total of 16 out of 21 articles report aims for replicating their study. Twelve articles specify how future replications should be differentiated. The most reported differentiation is to use different subjects in future studies. Replications with other research methodologies and replications in other domain and replications with other experimental designs are also identified in many articles. Four articles aim to replicate the study, but don't specify how to differentiate the replication.

Five articles don't report aims to study UML in other contexts [A4/A6/A8/A20]. Four out of these five articles that don't seek to conduct replications of their study are experience reports. This may be because of the nature of experience reports; they are often written in retrospect and are not so easy to replicate. Most of these articles have identified a need to conduct further refinements of their UML-experiences/method/language/tool under consideration to improve or make UML more precise.

Table 11: How the Authors' seek to differentiate their Study

Type of Data	Article IDs	No of Articles
Number of articles that suggest that further work should aim to study UML in different contexts.	A1, A2, A5, A7, A10, A11, A12, A13, A14, A15, A16, A19	12
Replicate with different subjects e.g. students or professionals.	A2, A5, A15, A16, A19	5
Replicate in other domains.	A1, A2, A7, A11, A12, A16	5
Replicate with an other research methodology, e.g. case study or experiment	A5, A16, A19, A7	4
Replicate with other experimental designs, e.g. different number of subjects or size and complexity of task.	A5, A13, A14, A16	4
Conducting the same analyses on data from existing experiments	A10	1

6.7.3 Further Refinements of the UML-Method under Consideration

Most of the articles that don't seek to replicate the study in other contexts, aim to refine their methods/tools/languages in future work. This need for further refinements is identified in the current work.

Ten articles report a need to refine the method/tool under consideration. These ten articles and the identified need for refinements are listed in Table 12.

Three out of these ten articles focus only on refinements in their aims for future work. The rest, seven articles, seek to both replicate the study in other contexts and refine the method under consideration.

Table 12: Articles that have Identified a need to Refine the Technology under Study

#AID	Refinements
A1	Refine the proposed tool. Incorporate other types of media and refine the transition between different levels of restriction.
A2	Study the precision of the Use Case Point method compared with expert estimates.
A3	Specify more precise descriptions of mappings.
A5	Investigate the optimal nesting level within the composite states.
A7	Develop guidelines on how to choose a type of stereotype appropriate for the purpose under consideration.
A8	Make the template language more readable, as well as extending support for it into our development environment.
A9	Improve existing guidelines
A18	Add specific heuristics to method under consideration to guide analysts in identifying, resolving and managing inconsistencies.
A20	Provide support for the user strategies observed and respond to subjects' suggestions by developing a help system for a step-by-step scenario generation procedure or a scenario template and providing domain-specific information with more examples. Finally, future work will develop an automatic scenario-generation tool with information extraction techniques
A21	Improve pattern language for Use Case descriptions.

6.7.4 Further Evaluation of the Cost-Effectiveness of UML

In this section I have collected data to answer the question whether authors focus on the cost-effectiveness of UML in their aims for future work.

Only two articles have addressed a need for further studies of cost-effectiveness of UML. One was the article on the use of OCL in UML models, which says, "future experiments should determine whether the benefits of devising OCL expressions justify their cost" [A17]. The other was the article on the use of Use Cases in estimation, which expresses a need to compare different methods for applying Use Cases in estimation with regards to precision of the estimates and the effort needed to

produce them [A2]. It does not seem to be much focus on investigating the cost-effectiveness of UML in the reported future work of the included articles.

6.7.5 Further Studies to Compare UML to Other Approaches

In this section I investigate what the authors of the included articles express about comparing UML to other approaches in their future work.

Three articles aim to compare UML to other approaches in future work. These three articles and their aims for comparing UML to other approaches are listed in Table 13.

The results from section 6.5 indicate that UML is not compared to other approaches to any extent. It does not seem that UML is going to be compared to other approaches in the future either, especially when it comes to "the basics" of UML. One study [A4] has developed a domain-specific language based on UML stereotypes and seeks to compare their use of stereotypes with approaches from other domains.

Table 13: Articles that aim to Compare UML to other Approaches.

#AID	Comparison of UML with other Approaches
A2	Study the precision of the Use Case Point method compared with expert estimates. Compare the different methods for Use Case estimation described with regards to precision of the estimates and the effort needed to produce them.
A4	Compare our use of stereotypes with approaches from other domains
A16	The method needs to be replicated and compared with, for example, usage-based testing.

6.7.6 Combine Approach under Study With Other Approaches

Three of the included articles aim to combine methods to investigate the possible effects.

One article [A21] wants to explore the relationship between the ability to predict the model of a finished application (Requirements pattern language) and the estimation of effort (Use Case Point method). One other article [A16] seeks to investigate a hybrid of two reading techniques, namely Usage-based reading and Checklist-based reading. Another article [A2] believes 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.

These articles are, however not concerned with combining UML directly to other approaches as these articles have another primary focus, namely a requirements pattern language that is applied to Use Case construction, an inspection technique that utilizes Use Cases and an estimation method that utilizes Use Cases.

6.7.7 Further Study of Other UML-Diagrams or Other Aspects than those under Consideration in Current study

One article [A19] considers investigating the rest of the modelling diagrams in UML, such as Activity Diagram, Sequence Diagram, and Statechart Diagram. The same article express that determining the core UML diagrams and the core constructs in each diagram are other interesting topics in the area.

6.7.8 A Broader Perspective for Future Work

In this section I investigate if the authors' see "the bigger picture", if their aims include investigating other use-areas of UML than those under study in the current article. I found two articles that expressed such needs, and these two articles were in relation to estimation and project management.

The first article [A2] states, "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 Cases 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 Cases estimate to provide a final estimate." The other article [A21] wants to explore the role of Use Cases in wider project management, such as the reporting of progress, and the management of requirements that changes.

6.7.9 What the Directions for Future Work Indicate About the Status of Current research

The authors have generally high ambitions for future work, as 76,2 percent of the included studies report that they intend to replicate their study. They seem to be aware of the need to replicate the study to support their findings. The fact that so many authors focus on replicating their studies, indicate that the body of evidence is far from large enough.

It appears that the main focus of future work lies with replication and refinement aspects. Almost as many articles aim to refine the technique under consideration as replicating the study. The need for replications of the studies is apparently there and most of the studies are not replicated. The fact that most of the articles also claim that refinements are needed indicate that more research needs to be done.

- 16 of the 21 included articles express that they intend to replicate the study, and 12 of these specify how to differentiate the replication.
- About half of the articles identify a need to refine the UML-method/tool under consideration.
- Only 5 articles don't mention anything about replicating the study. Out of these 5 articles, 4 articles are experience reports, and these are naturally not the easiest research methodology to replicate. The experience reports don't report on replications, but focus on refinements.
- One third of the articles have identified a need for both replicating and refining their UML-method/tool under study.

But what about other aspects of UML-usage that are interesting to evaluate, such as comparing UML to other approaches, evaluating cost-effectiveness or combining diagrams/methods? It doesn't seem to be much focus on such aspects.

The current research is not focused on comparing UML to other approaches to any extent; Only one article compare UML to other approaches, namely OPM. In directions for future work, three articles aim for comparing UML to other approaches.

One other question is if the use UML is cost-effective, that is yields economic benefits. Aims for evaluating the cost-effectiveness of UML in future work is reported in two articles. This indicate that researchers have ambitions for giving support of detailed use of UML, but they are not so engaged with the big questions attached to the use of MDD with UML, i.e. questions about when you should use the technique, to what extent and what benefits and costs it will entail.

The overall impression I have from the results of this sub-question is that research on Model-Driven Development with UML has a long way to go before a substantial body of knowledge can be collected. There are a great variety of topics in the included articles and the fact that the variety of studies is so large indicates that there is a need for replication of studies to collect evidence for the usefulness of UML.

7. Threats to Validity

The following Section discusses the most important threats to the validity of the results of this systematic review.

7.1 Choice of Journals and Conference Proceedings

The journals reviewed in this thesis were chosen because they are central in the field of Software Engineering. The conference proceeding UML/MODELS was chosen because it is the only conference that is concerned with the field of model-drivel development and UML. Hence, I believe that the journals and conference proceeding chosen constitute a proper representation of where empirical studies of MDD with UML are likely to be found. However, it is difficult to determine whether the selection of publication sources is representative of studies on Model-Driven Development with UML as these are spread over the field of Software Engineering research. The chosen publication sources are central, and I thus believe that the results are representative.

7.2 Selection of Articles

The systematic review of this thesis reviewed 21 articles selected from 963 articles from four publication sources. We were two persons that agreed upon the selection of articles. We did the identification of these articles individually and then came together to discuss the selection of articles. If there were disagreements, we discussed it and came to an agreement. We did a thorough job searching the publication sources for empirical studies of MDD with UML and we read through all articles that possibly could be of interest for our study.

It is often difficult to determine, based on the abstract, whether the article describes an empirical study. The abstract is supposed to summarize the work done in that specific article and it should be specified in the abstract how the research is validated. We used keywords, inclusion words, to help us decide whether an article reported MDD with UML based on its abstract. But based on my experience from this study I do not think it is possible to define definite keywords for deciding the empirical value of an article. The awareness of researchers when reporting their research doesn't seem to allow for this. It was sometimes difficult to decide based on the abstract if the article was interesting, and we ended up scanning through many complete articles to be able to decide the empirical value.

We may have missed articles during the selection of articles, due to our inexperience in the field of Model-Driven Development with UML and empirical methods, but I don't consider this to be a significant threat to the validity as we were very thorough in our search. This was the main reason for not having time to examine a larger set of publication sources; we used much time on reading and extracting data from the articles.

7.3 Data Extraction

During the analysis of the articles, we extracted data from 21 articles. The data provided answers to various questions of qualitative nature. The small number of included articles allowed us to read all the articles thoroughly and gather data in a thorough way.

It was occasionally difficult to extract data from the articles due to lack of a tradition for empirical research. It was not always obvious e.g. what the correct research method was for each study. Sometimes an article classified itself as a case study, while we would classify it as an experience report.

A common way of addressing this validity threat is to have data extraction performed independently by several reviewers so that the results can be compared and discussed [12]. We addressed this validity threat by just doing that; we individually read the whole article and agreed upon the extraction of data from each article. We did this for the articles in the pilot-study and continued to independently extract data from the articles until we were sure that we had a common idea of the type of data to be extracted from each article.

8. Conclusions and Future Work

Model-Driven Development (MDD) with UML is gaining widespread use in the IT-industry, and aims to raise the level of abstraction for software development by the use of models as key artefacts in software development, from system specification and analysis, to design and testing. The use of UML is claimed among others to improve the quality of software product deliverables, to support reuse and reduce the effort of developing and maintaining the software product. However, little is known with regards to the actual costs and benefits of MDD with UML.

This thesis is a systematic review of 21 selected articles that report empirical studies on Model-Driven Development (MDD) with UML. The objective of the investigation is to get an overview of the state-of-the-art for empirical studies of Model-Driven Development with UML, and to show the typical information found in articles of this topic. The data collected during analysis of these 21 articles, was used to address the following issues: The amount of empirical research of MDD with UML, the extent of different empirical research methods used, what different UML-diagrams that are studied and their benefits, for which application domains UML are studied, to what extent UML is compared to other approaches, the possibilities for meta-analysis and what the authors suggest about future research.

The main conclusion of this study is that it does not exist sufficient empirical evidence to draw conclusions regarding the usefulness of Model-Driven Development with UML. The few existing empirical studies of MDD with UML do mostly suggest that UML is useful, but they are too few and deal with too many aspects of UML-usage, to allow for definite conclusions about the usefulness of the technique.

We found relatively few studies, 2,2 percent, that empirically evaluate Model-Driven Development with UML in industrial projects or in experiments with human subjects. Research on MDD with UML has so far had little focus on empirical studies, and empirical studies have seldom MDD with UML as a topic. The empirical studies that involve UML usage have also often another primary focus on e.g. inspection of software artefacts.

Further, the existing studies deal with very many different aspects of UML usage. This makes it difficult to arrive at a conclusion regarding how to use UML and regards to utilitarian value and costs based on empirical studies. It was, however, a strong increase of empirical studies of MDD with UML in 2005, so if this represents a general trend, the possibilities for making such conclusions may be improved.

Experiments are the most used research method within studies of MDD with UML, and this study found four times as many experiments as case studies. In addition to case studies, it was also a number of experience reports. These also report experiences with UML usage in industry projects, but with less formality than the case studies.

The Conference proceeding UML/MODELS, as the most important publication source for research of MDD with UML studied in this thesis, had primarily case studies and

experience reports among the empirical studies. The most prestigious publication source that was examined in this selection, TSE, had only experiments, indeed only two. This shows that experiments have a strong position among scientists that make use of empirical methods, and could also be considered to be the research method that gives the best and most solid results. At the same time, it appears that those who study UML are most interested in studies that report industrial experiences, i.e. case studies and experience reports. The research is very much focused on formulating and proposing new methods and extensions to UML, and not so much in evaluating existing approaches.

This study separates the studies in accordance to how UML is used in relation to comprehension of a system, construction of a system, correctness of a system and predictability in Software Engineering. The main results show that UML usage could yield importance for the results of all these aspects, and all the aspects have improvement potential when UML is used. However, some of the articles study the interaction between use of UML, experience and competence of the developers and application domain. These studies show that these factors are decisive for successful use of UML and that it often has as much importance as the technology itself for how well the results become. This indicates that future studies of MDD with UML should describe the experience and competence of developers, and also the domain where UML is used. Further, it is especially a need for more studies of what level of competence that are necessary to utilize different aspects of UML, and how different aspects of UML are adapted to different application domains.

This study looked especially at what the researchers behind the studies viewed as important future work. Most of them found it necessary to replicate the study, perhaps with another type of subjects. Others found it necessary to refine the UML-based technique under study. Only two of the studies argued that future studies should evaluate the cost-effectiveness of UML. This indicate that researchers have ambitions for giving support of detailed use of UML, but they are not so engaged with the big questions attached to the use of MDD with UML, i.e. questions about when you should use the technique, to what extent and what benefits and costs it will entail. Hence, the results could be useful for companies that already make extensive use of UML, but these results do not provide much guidance when UML is introduced in a company.

Future Work

This review has shown that it exists relatively few studies that empirically evaluate Model-Driven Development with UML in industrial projects or in experiments with human subjects.

A proposal for future work would be to replicate this study, but to include a larger set of publication sources. Empirical studies of MDD with UML are spread over the whole Software Engineering field, and it may be necessary to search a larger set of publication sources to get enough articles to be able to perform meta-analytical procedures. We had not time available to include a larger set of publication sources in this thesis as the selection of articles and extraction of data was too time consuming.

This review showed a strong increase of empirical studies of MDD with UML in 2005. A proposal for future work could be to include the following year to investigate if this is a general trend, and if it were, it would give opportunities to make improved conclusions about the usefulness of MDD with UML.

Furthermore, a proposal for future work is to investigate other aspects of empirical studies on Model-Driven Development with UML that were not taken into account in this study. We extracted more data from the articles than we got use for in this thesis. This data was mainly focused on what type of subjects that were used, tasks, collection of data and analysis, and hypothesis etc.

References

1. [cited-3.1.2007]; Available from: http://www.eclipse.org/proposals/eclipse-mddi/.
2. [cited-5.1.2007];-Available-from: http://www.omg.org/gettingstarted/what_is_uml.htm.
3. [cited-16.01.2007];Available from: http://www.sdl-forum.org/SDL/index.htm.
4. [cited-13.01.2007]; Available from: http://www.iec.org/online/tutorials/sdl/.
5. [cited-15.01.2007];Available-from: https://users.cs.jmu.edu/tuckerrj/Courses/Cs555-Common/StudyUnits/Unit5/SASD/sld001.htm.
6. [cited12.01.2007];Available-from: http://www.excelsoftware.com/sasdtopic.html.
7. [cited-15.01.2007;Available-from: http://www.meta-analysis.com/pages/why_do.html.
8. Briand, L., et al., *Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions.* Empirical Software Engineering, 1999. **4**(4): p. 387-404.
9. Glass, R.L., I. Vessey, and V. Ramesh, *Research in Software Engineering: An analysis of the literature* Information and Software Technology, 2002. **44**(8): p. 491-506.
10. Hayes, W. *Research synthesis in software engineering: a case for meta-analysis*. in *Proceedings of the International Symposium on Software Metrics*. 1999.
11. Holt, N.E., *A systematic review of case studies in software engineering*. 2006, University of Oslo. p. 1-82.
12. Kitchenham, B. *Procedures for Performing Systematic Reviews*. 2004 [cited; Available from: http://www.idi.ntnu.no/emner/empse/papers/kitchenham_2004.pdf.
13. Kitchenham, B., L. Pickard, and S.L. Pfleeger, *Case Studies for Method and Tool Evaluation* IEEE Software 1995: p. 52-62.
14. Kruchten, P., *The Rational Unified Process An Introduction Third Edition* 2003: Addison-Wesley.
15. Miller, J., *Applying meta-analytical procedures to software engineering experiments.* The Journal of Systems and Software 2000(54): p. 29-39.
16. Pickard, L.M., B. Kitchenham, and P.W. Jones, *Combining Empirical Results in Software Engineering.* Information and Software Technology 1998. **40**: p. 811-821.
17. Rumbaugh, J., I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual, Second Edition*. 2005: Addison Wesley.
18. Segal, J., A. Grinyer, and H. Sharp. *The type of evidence produced by empirical software engineers*. in *Proceedings of the 2005 workshop on Realising evidence-based software engineering*. 2005. St. Louis, Missouri: ACM Press.
19. Sjøberg, D.I.K., et al., *A Survey of Controlled Experiments in Software Engineering.* IEEE Transactions on Software Engineering, 2005. **31**(9): p. 1-21.

20. Wohlin, C., M. Höst, and K. Henningson. *Empirical Research Methods in Software Engineering*. in *ESERNET 2001-2003*. 2003. LNCS 2765.
21. Yin, R.K., *Case Study Research Design and Methods*. 2003: Sage Publications.
22. Zelkowitz, M.V. and D. Wallace, *Experimental Validation in Software Engineering.* Information and Software Technology, 1997. **39**(11): p. 735-743.

Appendix A - Data Extracted from Articles

The conference proceeding UML/MODELS:

Article ID	**A1**
Title	Supporting Several Levels of Restriction in the UML
Author	Christian Heide Damm, Klaus Marius Hansen, Michael Thomsen, Michael Tyrsted
Year	2000
Source	UML 2000, LNCS 1939, pp. 396-409, 2000
Type of study	Experience report.
Intent	Reports on experiences from investigating the practice of initial, creative, and collaborative problem domain modelling through a number of concrete projects. Argue for the usefulness of several levels of restriction of the UML meta model for the usefulness of transferring a model element from one level of restriction to another. Present and discuss a tool, Knight, which illustrates the main ideas.
Results/ Lessons learned.	In an object-oriented software development project – and in the process of creating a UML model – the UML is *not* sufficiently flexible, and hence it does not support all phases of object-oriented modelling. Likewise, existing tools do not support initial modelling phases in a satisfactory way. UML is often too restrictive in initial, informal, and creative modelling, and it is in some cases not restrictive enough, e.g., for code generation. UML should support the initial phases of modelling, where incomplete diagram elements and freehand drawing are heavily used. It is necessary to do more expressive, but less restrictive, modelling in the initial and creative problem domain modelling. The fact that the UML metamodel disallows practices used in modelling suggests that the UML metamodel should be modified. Instead of having several more or less incidental metamodel, the UML metamodel itself could contain different *levels*. The levels of the UML metamodel should cover the range from very expressive models to models close to code. A change in the UML metamodel should be accompanied by appropriate tool support. Modelling involves going from more or less expressive diagrams to more restricted models, and tools should support this process.
Future work	- CASE tools are too concerned with software engineering aspects of development. Instead, aspects such as creativity, flexibility, and idea generation should be focused on and more widely supported. - Set up longitudinal studies of Knight in use in companies.

	- Refinements are needed in the Knight tool. - Provide guidance or automation of a tool. - Experiment with the transportable Mimio technology. - Commercialize the tool.
Development phase	Initial, creative, and collaborative problem domain modelling.
Application domain	A large, globally distributed shipping company/a global customer service system for the company. An Embedded control system for flow meters.
Duration	
Project/ Participants Details	Project Dragon: Involved a research group and a large, globally distributed shipping company. The goal of the project was to implement a prototype of a global customer service system for the company. This was realized over a one and a half year period by the development of a series of successful prototypes. In this project, three of the authors participated actively and observed ongoing work. Project Danfoss: was concerned with implementing an embedded control system for flow meters. The project lasted a year and involved experienced developers from a research group and engineers from a private company. One of the authors participated in this project. This involved formal observations of work and active participation. Thus our approach was a mix of (ethnographic) observations and active involvement in the project studied Both projects used an iterative object-oriented approach to system development. Throughout development, UML was used on whiteboards and in CASE tools to visualise an emerging understanding of the problem and solution domains.
Collection of data and analysis	Experiences from two projects. Project 1: Three of the authors participated actively and observed ongoing work. Project 2: One of the authors participated in this project. This involved formal observations of work and active participation. Thus our approach was a mix of (ethnographic) observations and active involvement in the project studied Three representative scenarios distilled from the studies, used as a basis for analysis of current modelling practice.

Article ID	**A2**
Title	Estimating Software Development Effort Based on Use Cases – Experiences from Industry
Author	Bente Anda, Hege Dreiem, Dag I.K. Sjøberg, Magne Jørgensen
Year	2001
Source	UML 2001, LNCS 2185, pp. 487-502, 2001
Type of study	Case study
Intent	Evaluate the application of a method for effort estimation based on Use Cases points. Compare estimates based on Use Cases Points for three development projects with estimates obtained by experts, in this case senior members of the development projects, and actual effort.
Result	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. 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. 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. - The use of included and extending Use Cases. - 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.
Future work	- Study the precision of the Use Case Point method compared with expert estimates. - Study the precision of the estimates when using the Use Case Points method in different types of projects and with estimators that have different levels of experience. - Combine 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. - Activities in a development project that do not depend on size or Use Case Points should be estimated in alternative ways and then be added to the Use Case estimate to provide a final estimate. - Compare the different methods for Use Case estimation with regards to precision of the estimates and the effort needed to produce them. - Investigate whether other methods for Use Case estimation are suitable for Use Case models with less detail.
Development phase	Estimation in requirements engineering process.

Application domain	E-commerce and call-centres, in particular within banking and finance.
Study design	3 projects. Projects A, B and C. The 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.
Location	The 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. 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.
Duration	7 months.
Case tools	Programming environment: Java (Visual Café and JBuilder), Web Logic, MS Visual Studio, Java (Jbuilder)
Software artefacts	Project A: Use Case model, iteration plan and spreadsheets with estimates and effort. Project B: Requirements specification with Use Case diagrams and textual descriptions of Use Cases, project plan and time sheets recording the hours worked on the project. Project C: 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.
Participant detail	6+6+9 (=21) subjects. Professionals. 0-17 years of experience.
Collection of data and analysis	We compared estimates based on Use Case Points with estimates obtained by experts and actual effort. 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, 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. Data from project C was collected from project documents, and from an interview with two of the project members.

Article ID	**A3**
Title	Practical Experiences in the Application of MDA
Author	Miguel de Miguel, Jean Jourdan, and Serge Salicki
Year	2002
Source	UML 2002, LNCS 2460, pp. 128-139, 2002. Springer-Verlag Berlin Heidelberg 2002
Type	Experience report
Intent	Introduce some problems detected in the process of adoption of MDA solutions in Thales.
Result	The adoption of MDA require to take some decisions about the technologies to be used, and the tools and methods that can support MDA. These problems can create problems equal or more complex than the problems that MDA try to solve. If there is not a precise description of mappings, two different implementation of mappings can generate very different code or models and this can create the dependencies between the software and the mapping solution used
Future work	OMG has started to specify more precise descriptions of mappings.
Development phase	
Application domain	Many different domains.
Duration	
Project/ Participant Detail	During last years, Thales has started some actions for the adoption of model driven engineering techniques. Two pilot programs have UML and modelling engineering as main references in the software development process. These pilot programs include the application and evaluation of these techniques in four real projects, and other actions develop supports for the adoption of these techniques and for their practical application in the near future. The real projects use current solutions for the transformation of models and code generation and evaluate the cost of their application. And the innovation actions include the development of UML extensions to support specific domains modelling languages, techniques, and technologies, and the models transformations [25].
Collection of data and analysis	

Article ID	**A4**
Title	Using UML for Information Modelling in Industrial Systems with Multiple Hierarchies
Author	Peter Fröhlich, Zaijun Hu, and Manfred Schoelzke
Year	2002
Source	UML 2002, LNCS 2460, pp. 63-72, 2002
Type	Experience report
Intent	Report on experiences in applying a UML domain-specific language for information modelling of industrial plant applications, which typically consist of multiple structural hierarchies. - Introduce a meta-model, which describes, how the information models can be expressed in UML. - Discuss a simple case study, which applies this model in the context of ABB`s Industrial IT platform. - Describe our experiences with UML-based modeling in this domain and discuss the differences between a UML-based representation and the concepts of IEC 61346.
Result/ Lessons learned	We have modelled two larger applications in the field of control system engineering successfully with the domain-specific models. Each of these applications was easily mapped to our Industrial IT platform. The teams creating these models consisted of engineers as well as computer scientists. We found that the usage of our UML models helped both groups. They explained to the computer scientists the idea of multiple hierarchies and helped the engineers to phrase their ideas within the concepts, the Industrial IT platform supports. Earlier attempts to map UML models representing one hierarchy (e.g. only the functional hierarchy) into the platform and merge them with structures for the other hierarchies later were not successful. We therefore believe, that the complexity of the multiple hierarchies can only be addressed in a model, which describes all hierarchies in parallel. In our experience, such a model can only be created at the beginning of the system development or application integration project. This means, at the beginning of the application integration process a set of object types with their relations in the different structures are created in the form of a library of object types. The attributes within one application aspect can then still be adapted to changes during the project, and new applications can be added with little effort. In contrast to our previous work [11], where we used a formal meta-model with a custom semantics for modelling industrial applications, the current work is based on UML stereotypes. This more informal use of a meta model has been better accepted by the users than our previous

	logics-based approach.
Future work	Part of our future work is to compare our use of stereotypes with approaches from other domains
Development phase	
Application domain	Industrial plant applications with multiple hierarchies. Control system engineering.
Duration	
Project/ Participant detail	Computer scientists and engineers.
Collection of data and analysis	

Article ID	**A5**
Title	Evaluating the Effect of Composite States on the Understandability of UML Statechart Diagrams
Author	José A. Cruz-Lemus, Marcela Genero, M. Esperanza Manso and Mario Piattini
Year	2005
Source	MODELS 2005, LNCS 3713, pp. 113-125, 2005
Type of study	Experiment
Intent	Evaluate the effect of composite states on the understandability of the UML statechart diagrams.
Hypotheses	H0: the use of composite states does not improve the understandability efficiency of an UML statechart diagram. H1: the use of composite states improves the understandability efficiency of an UML statechart diagram.
Result	The results of the experiment confirm, to some extent, our intuition that the use of composite states improves the understandability of the diagrams, so long as the subjects of the experiment have had some previous experience in using them.
Future work	- Replicate with experienced practitioners, as well as by taking data from real projects and using other experimental design. - Investigate the optimal nesting level within the composite states. - Check validity against UML2.0 Meta-model. - In addition, we will investigate whether our proposed metrics [2] could be used as maintainability indicators of UML statechart diagrams.
Development phase	OO design. And code generation.
Application domain	Experimental task: an ATM machine and a phone call.
Study design.	Controlled experiment. Replication.
Location	Exp. 1: University of Murcia. Exp. 2: University of Alicante.
Exp year	Exp. 1: February 2005, Exp. 2: March 2005.
Duration	
Case tools	
Software artefact	UML statechart diagrams. Composite states.
Participant detail	Exp. 1: 55 Computer Science students from the University of Murcia. All the subjects were in the fourth year of Computer Science and had received a complete Software Engineering course in which they had studied modelling techniques, including UML. They also received a short training session before the performance of the experiment.

	Exp. 2: 178 subjects. Replication. The subjects were Computer Science students. Got extra points in exam to participate, but it was voluntarily to participate. The skill of the subjects using UML for modelling, especially UML statechart diagrams, was much lower in this replication, as most of them had only a few months of experience, and they had not worked with some UML meta-model constructs (e.g. composite states) yet. They received the same training session as in the original experiment before performing the replication, but even with this, their experience level was much lower, compared to the first group of subjects.
Collection of data and analysis	Questionnaire. Time logging.

Article ID	**A6**
Title	Model-Driven Engineering in a Large Industrial Context — Motorola Case Study
Author	Paul Baker, Shiou Loh, and Frank Weil
Year	2005
Source	MODELS 2005, LNCS 3713, pp. 476–491, 2005. c Springer-Verlag Berlin Heidelberg 2005
Type	Experience report.
Intent	Present experiences within Motorola in deploying a top-down approach to MDE for more than 15 years.
Results/ Lessons learned	We have found that through the coordinated and controlled introduction of MDE techniques, significant quality and productivity gains can be consistently achieved, and the issues encountered can be handled in a systematic way. - Code generators are much better than humans at finding optimal and correct sequences of bit-manipulation instructions for performing the marshalling and dealing with data from different endian machines. -Through the coordinated and controlled introduction of MDE techniques, significant quality and productivity gains can be consistently achieved, and the issues encountered can be handled in a systematic way. - Field data has shown that code generated by Mousetrap has fewer defects than hand code or code generated by vendor tools. - Architects and designers were reluctant to invest the extra effort needed to develop rigorous models since the benefit of automated test generation did not immediately justify the extra effort within their project scope. - Users often did not understand the differences between test generation and test specification.... and hence do not gain the full benefits - The use of scenario-based test generation tools yields an approximately 33% reduction in the effort required to develop test cases. - Motorola has seen tremendous gains in some areas of the development process. This reduction is attributed to the ability to add a model test that illustrates the problem, fix the problem at the model level, test the fix by running a full regression test suite on the model itself, regenerate the code from scratch, and run the same regression test suite on the generated code.
Future work	Continue to invest in MDE technology improvement ranging from automation technologies, metrics, profile development, meta-modeling, and analysis tools.
Application Domain	Telecommunication.
Development	

phase	
Duration	20 years.
Project/ Participant details	Motorola has been active in MDE for nearly two decades and has seen incredible successes and glaring failures.
Collection of data and analysis	

Article ID	**A7**
Title	Properties of Stereotypes from the Perspective of Their Role in Designs
Author	Miroslaw Staron, Ludwik Kuzniarz
Year	2005
Source	MODELS 2005, LNCS 3713, pp. 201-216, 2005. c Springer-Verlag Berlin Heidelberg 2005
Type of study	Quasi experiment.
Intent	Evaluate whether stereotypes are appropriate for the purpose/quality of stereotypes.
Hypotheses	How to elaborate quality assessment criteria for new stereotypes based on existing stereotypes which are known to be "good"?
Result	The results indicate that there exist some relationships between different categories in these classifications which make certain stereotypes (decorative) not usable for a certain purpose (code generation). The set of standard UML constructs is known to be insufficient for all purposes and the users of UML often create stereotypes to enrich their set of modelling elements. Four categories of stereotypes according to expressiveness: Decorative stereotypes, Descriptive stereotypes, Restrictive stereotypes and Redefining stereotype The classification organizes the stereotypes into three categories according to their role: Code Generation stereotypes, Virtual Metamodel Extension stereotypes, Model Simplification stereotypes.
Future work	Develop guidelines on how to choose a type of stereotype appropriate for the purpose under consideration. Use properties in our further research for developing guidelines for creating and using stereotypes in a more efficient way. Validate the method in a company.
Development phase	
Application domain	
Study design	Before classifying all stereotypes, the appropriateness of the classifier was verified by comparing the classifier's classification results to classifications of other subjects (Auxiliary Experiment) and was performed in an academic environment with doctoral students classifying a subset of thirteen stereotypes
Location	
Duration	
Case tools	
Software	Stereotypes.

artefact	
Participant detail	Classifier. The auxiliary study was conducted with two additional subjects. The subjects (and the classifier) possessed the necessary knowledge of stereotypes and they also participated in other studies on stereotypes. They were sufficiently experienced in modeling, object orientation and programming.
Collection of data and analysis	We identified the types of a set of 98 stereotypes and their properties in an empirical way by investigating stereotypes from UML profiles used in industrial software development. In the criteria elaboration we group stereotypes using categories from two classifications of stereotypes, (i) according to their role and (iii) according to their expressiveness. Based on the results of classifying stereotypes we elaborated types of stereotypes. The properties are intended to be used in further research for developing guidelines for creating and using stereotypes in a more efficient way.

Article ID	**A8**
Title	Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service-Oriented Application — Experiences and Challenges
Author	Marek Vokáč and Jens M. Glattetre
Year	2005
Source	Models 2005, LNCS 3713, pp. 492- 506, 2005. c Springer-Verlag Berlin Heidelberg 2005
Type	Experience report
Intent	Report of experiences of a re-implementing of a commercial Customer Relationship Management application, in stages, as a service-oriented, multi-tier application with a domain-specific UML language.
Lessons learned	Perhaps the single most positive consequence of using a model is to raise the general consciousness level about the need for well-designed, thought through interfaces. Standardization is also an important benefit. - The combination of modelling and generation helps by codifying and enforcing the "standard" way of doing things. Simultaneous generation of remote interfaces, local implementations, data and message contracts as well as unit test skeletons and documentation pages from a single model ensures that all of these artefacts are actually created - We have successfully used this approach on a prototype scale, and are now transitioning to full-scale development. - In general, increased modelling experience correlates with an increased perception of the benefits of the approach. - Current modelling tools based on UML reflect the fact that UML semantics are informal, while specific enough to point clearly in the direction of an object oriented target language. - Tools that support the creation of Domain-Specific Languages are not yet ready for heavy industrial use. - Our solution has been to create a simple code generator based on text substitution, and to use a UML Profile to define the additional semantics we need in the modelling language. - Our experience so far is that the approach works quite well for those lower layers of the application that express similar functionality repeatedly, such as Data Access Objects for individual tables. When modelling more high-level services, the emphasis is more on the standardization of naming and behaviour, and the generation of skeletons rather than complete functionality.
Future work	- Making the template language more readable, as well as extending support for it into our development environment. - Further major development will probably wait for the availability of more sophisticated tools. We continually strive to find the correct balance between investment in inhouse tools and dependence on external tools. In the future, a switch to externally developed tools is quite

	probable, when sufficiently mature and well-supported tools are offered.
Development phase	Domain specific language, applies all phases. This paper focuses on a code generation tool.
Application domain	A commercial Customer Relationship Management application /a service-oriented, multi-tier application.
Duration	
Participant detail	Six developers. The developers' experience with modelling ranges from minimal to extensive (more than 5 years), and their time with the company from 7 years down to just a few months.
Collection of data and analysis	- Conducting interviews with architects and developers in our organization to extract the knowledge presented here. - In order to define our DSL, we have taken the minimum set of concepts needed to capture our modelling requirements, and translated them into a UML profile.

The Empirical Software Engineering Journal:

Article ID	**A9**
Title	Replicating the CREWS Use Case Authoring Guidelines Experiment
Author	Karl Cox, Keith Phalp
Year	2000
Source	Empirical Software Engineering, 5, 245–267 (2000)
Type of study	Experiment
Intent	Replicate a previous experiment on evaluating the effects of using a set of Use Case authoring guidelines (the Crews guidelines).
Hypotheses	3 hypotheses on the effects of content guidelines and 4 hypotheses on the effects of style guidelines on the Use Cases.
Result	The use of content guidelines led to more complete Use Cases. The use of the style guidelines together with content guidelines had some positive effect on the structure of the Use Cases, but was not useful for Use Case content. Use of only style guidelines had a negative effect on both aspects of Use Case quality. Many of the content guidelines were applied "by chance" also by the subjects not following the guidelines, hence the actual guidelines were not very useful. the CREWS guidelines do not necessarily improve the use-case descriptions
Future work	Further experiments on the effects of the guidelines. Development of guidelines that help analysts locate appropriate information in documents for use in use-case descriptions. Experiments that combine both reading and writing guidelines.
Development phase	Requirements engineering.
Application domain	Experiment task: Supermarket checkout
Study design	A: A control group in which subjects were given the problem statement only B: An experimental group in which the subjects were given the problem statement and CREWS Style Guidelines C: An experimental group in which the subjects were given the problem statement and CREWS Content Guidelines D:An experimental group in which the subjects were given the problem statement and CREWS Style and Content Guidelines Groups A and B contained four subjects each and groups C and D three subjects each. Pen and paper with textual description and guidelines.
Location	Bournemouth University
Exp year	
Duration	One hour to write the Use Cases.
Case tools	None.

Software artefact	Use Case.
Participant details	14 subjects, full-time students at a Masters degree course in Software Engineering. The subjects received a half-day seminar on Use Cases a week before the experiment. This was their first exposure to Use Cases. The seminar included a discussion of use-case structure and contents, such as levels of abstraction. The subjects also completed a form about their experience of systems analysis and programming. Experience ranged from 0 to 6 years in systems analysis and from 0 to 15 years in programming. Ages ranged from 23 to 48.
Collection of data and analysis	The subjects received instructions (including Use Case guidelines for some of the experimental groups). The resulting Use Cases were evaluated against a set of criteria defined by the authors.

Article ID	**A10**
Title	Are the Perspectives Really Different? – Further Experimentation on Scenario-Based Reading of Requirements.
Author	Björn Regnell, Per Runeson, Thomas Thelin.
Year	2000
Source	Empirical Software Engineering, 5, 331–356, 2000.
Type of study	Experiment
Intent	1) To investigate if different perspectives taken by code inspectors (using the inspection technique Perspective-Based Reading, PBR) detect different defects and if one perspective is superior to the others. 2) To compare PBR to checklist-based inspection. Partial replication of previous study.
Hypotheses	Heff: The perspectives are different with respect to efficiency measured in terms of the number of defects found per hour of inspection. Hrate: The perspectives different with respect to effectiveness or detection rates measured in terms of the fraction of defects identified. Hfound: The perspectives are different with respect to defects detected measured in terms of the distribution of defects found.
Result	The analysis result show that (1) there is no significant difference between the three perspectives in terms of defect detection rate and number of defects found per hour, (2) there is no significant difference in the defect coverage of the three perspectives, and (3) PhD students with a checklist approach find significantly more defects per hour and have significantly higher detection rate than MSc students with PBR approach. The results suggest that a combination of multiple perspectives may not give higher coverage of the defects compared to single-perspective reading. It is also indicated that individual abilities and motivation are more important than the reading technique used.
Future work	Conduct the same analyses on data from existing experiments as well as new replications with the purpose of evaluating differences among perspectives.
Development phase	Inspection, applies to all development phases.
Application domain	Experimental domain: ATM and Parking Garage
Study design	Partial replication of previous studies. Control group. Formal factorial experiment. 3 groups, 10 subjects per perspective.
Location	
Exp year	Spring 1998
Duration	
Case tools	None.

Software artefacts	Two requirements documents and scenarios for three perspectives (*user* applying Use Case modelling, *designer* applying structured analysis, and *tester* applying equivalence partitioning). Reporting templates for time and defects.
Participant details	30 Msc and PhD students + a control group of 9 PhD students used a checklist reading technique. The subjects are fourth-year students at the Master's programme in Computer Science & Engineering and Electrical Engineering at Lund University and PhD students at the department of Communication Systems and the Department of Computer Science at the same university. The MSc students were all given a two hour introduction, while the PhD students were given a one-hour introduction. An overview of the study was given together with a description of the defect classification. The MSc students practised their own perspective reading technique. The data collection forms were also explained and used during the exercise. The checklist method was described for the PhD students.
Collection of data and analysis	The subject's defects' logs.

Article ID	**A11**
Title	Empirical Evaluation of CASE Tools Usage at Nokia
Author	Maccari, Riva
Year	2000
Source	Empirical Software Engineering, 5, 287–299 (2000)
Type	Structured questionnaire
Intent	Evaluation of case tool usage at Nokia, understanding the domains and consequences of CASE tools
Result	From this survey, it emerged that CASE tools support is reputed most useful for the following functions: graphical drawing, automatic documentation generation and storage of diagrams. The results hint to a mismatch between the features required by the developers and those offered by CASE products. No feature has been rated ***very well implemented.*** This may mean that, in general, commercial CASE tools fail to implement features in an optimal way. Modelling for standard UML-notation, be able to edit all the UML diagrams, perform diagrams and support for design specification is by the respondents considered to be sufficiently well implemented. - Sufficiently well implemented and highly desired: Support for standard UML notation, Perform diagram analysis (e.g. consistency check), Support design specification, Utilise a repository, Allow easy editing of text notes inside diagrams, Allow easy editing of graphical data (diagrams) - Insufficiently well implemented but highly desired: Be intuitive and easy to use, Automatically generate well-structured documents, Manage versioning
Future work	- Investigate the reasons that lie behind results. Two evident examples are the lower-than-expected usefulness of code generation and the expressed need for configuration management and testing related features. - Investigate the consequences of CASE tools usage in our software development organisation using an interpretive approach. - Carry out similar experiments with developers and teams belonging to different telecom industries.
Development phase	All phases.
Application domain	Telecom software/Mobile handsets.
Study design	Interpretive. Context: It was performed on the context of various mobile phone software development units inside Nokia, both in Europe and in the US.

Location	Finland (6), Europe (7) and US (1)
Exp year	Fall 1999
Duration	
Case tools	Respondents had experience with and rated these case tools: Rose98, objectiveTime developer, Prosa/om, Rhadsody, Qlm 2.1
Software artefact	
Participant details	14 design engineers and software developers. The respondents have between 6 months and 25 years of software development experience, with an average of 8.25 years of experience. Their experience in software development inside Nokia ranges between 6 months and 7 years, with 2.71 years on average. Their experience in mobile phones software development ranges between the same value of 6 months and 7 years, but with an average of 2.14 years.
Collection of data and analysis	A Structured questionnaire was sent to 48 design engineers and software developers by electronic mail during August 1999. The answers were collected and analysed during fall 1999. We received 14 responses, for a response rate of almost 30%. According to Edwards (1972), as quoted in Wood (1999), a response rate of 20% to 30% for mailed questionnaires is considered acceptable. We have analysed the collected data using two simple descriptive statistics: median and standard deviation.

Article ID	**A12**
Title	An Initial Experimental Assessment of the Dynamic Modelling in UML
Author	MARI CARMEN OTERO, JOSE´ JAVIER DOLADO
Year	2002
Source	Empirical Software Engineering, 7, 27–47, 2002.
Type of study	Experiment
Intent	The goal of this empirical study is to compare the semantic comprehension of three different notations for representing the dynamic behaviour in unified modeling language (UML): (a) sequence diagrams, (b) collaboration diagrams, and (c) state diagrams.
Hypotheses	H1: There is no difference between 3x3=9 experimental conditions with respect to . . . H2: There is no difference between the subjects using the three diagram types with respect to . . . H3: There is no difference between the subjects reading OO design documents with respect to . . .
Result	The main conclusion of this study is that the comprehension of the dynamic modelling in object-oriented designs depends on the diagram type and on the complexity of the document. The software project design written in the UML notation is more comprehensible, when the dynamic behaviour is modelled in a sequence diagram. While if it is implemented using a collaboration diagram, the design turns out to be less comprehensible as the application domain, and consequently, the document is more complex.
Future work	Include the effects of the interactions. We have shown in this experiment that investigating the interaction between factors is essential to understanding the results of the experiment. More practical work with the models is needed, in order to identify which diagrams provide the most appropriate semantics for each domain.
Development phase	Design.
Application domain	Experimental task: A Simple Cellular Telephone, a Library System and a Digital Dictaphone
Study design	Goal-question-metric (GQM) for organizing the experiment. 3x3 factorial design with repeated measures. Blocking technique.
Location	
Exp year	May 1999.
Duration	
Case tools	None.
Software artifacts	Sequence, collaboration and state diagrams.

Participant details	18 Last year students. The students attended classes about UML regularly before the experiment and participated in a training exercise (1 day) related to the correct use of the material.
Collection of data and analysis	Chronometer. Questionnaires were scored. Collect time.

ArticleID	**A13**
Title	Evaluation of Usage-Based Reading - Conclusions after Three Experiments
Author	Thomas Thelin, Per Runeson, Claes Wohlin, Thomas Olsson, Carina Andersson.
Year	2004
Source	Empirical Software Engineering, Volume 9, Issue 1 - 2, Mar 2004, Page 77
Type of study	Experiment
Intent	Evaluate the usage-based reading, UBR, technique. Exp. 1: Compare Use Case driven inspections with prioritized Use Cases versus randomly ordered Use Cases. Exp. 2: Compare UBR with checklist-based reading (CBR). Exp. 3: Investigate whether the reviewers perform better in the preparation phase if they develop Use Cases as part of the inspection or if it is better to utilize pre-developed Use Cases in the inspection. The third experiment also studies UBR as part of the inspection process.
Hypotheses	**Exp. 1:** Is UBR effective in finding the most critical faults? Is UBR efficient in terms of total number of critical found faults per hour? Are different faults detected using different priority orders of Use Cases? **Exp. 2:** Is UBR more effective and efficient than CBR? **Exp. 3:** Is pre-developed Use Cases needed for UBR? HEff - There is a difference in efficiency (i.e., found faults per hour) between the reviewers utilizing pre-developed (Util) Use Cases and the reviewers who develop Use Cases (Dev). HRate - There is a difference in effectiveness (i.e., rate of faults found) between the reviewers utilizing pre-developed Use Cases and the reviewers who develop Use Cases. HFault - The reviewers utilizing pre-developed Use Cases detect different faults than the reviewers who develop Use Cases.
Result	The **main results** are (1) UBR is an efficient and effective reading technique that can be used for user-focused software inspections, (2) UBR is more efficient and effective if the information used for UBR is developed prior to, instead of during the individual preparation, and (3) the meeting affects the UBR inspection in terms of increased effectiveness and decreased efficiency. From the **first experiment**, it is concluded that the reviewers find different faults using prioritized Use Cases compared to randomly ordered Use Cases. Furthermore, UBR (prioritized Use Cases) is significantly more effective and efficient than randomly ordered Use Cases in finding faults of high importance for a user. From the **second experiment**, it is concluded that UBR is significantly better than CBR in terms of both effectiveness and efficiency in finding the faults that affect the user the most. The results show that reviewers

	applying UBR are more efficient and effective in detecting the most critical faults from a user's point of view than reviewers using CBR. From the **third experiment**, it is concluded that is more efficient to use pre-developed Use Cases for UBR. However, there is a trade-off of whether the Use Cases should be developed beforehand or on the fly during inspection. The meeting of the third experiment increased the effectiveness of the faults found, but decreased the efficiency.
Future work	The series of experiments presented in this paper shows that UBR has the potential to become an important reading technique. However, more research is needed and there are several areas that should be considered in order to further develop and investigate UBR. Among these are replications, time-controlled reading (add time limits to the Use Cases), reading techniques for inspection meetings and case studies in software organizations.
Development phase	Inspection, applies to all development phases.
Application domain	Experimental task: Taxi system.
Study design	Exp. 3: Two groups. Controlled variable: experience.
Location	Sweden (Lund University, Blekinge Institute of Technology).
Exp. year	Exp. 1: fall 2000. Exp. 2: spring 2001.Exp. 3: Fall 2001.
Duration	Exp. 3: Two days.
Case tools	None.
Software artefacts	Exp. 1&2: High-level design (Use Cases). Exp. 3: Textual requirements document (natural language), Use Case documents, A design document. In addition, the communication between the system and the users is specified. Furthermore, the design document contains two message sequence charts (MSC)
Participant details	Exp. 1: 27 Students of their third year of the software engineering Bachelor's program at Lund University (Campus Helsingborg). Exp. 2: 23 Students of their fourth year of the software engineering Master's program at Blekinge Institute of Technology. Exp. 3: 82 (34+48) Students. The experiment was a mandatory part of two courses in verification and validation. The subjects in Hbg were 34 third-year students at the software engineering Bachelor's program at Lund University. The students were almost finished with their education and have experience in requirements engineering, Use Case development, software design and in the particular application domain (taxi management systems). The subjects in Rb were 48 fourth-year software engineering Master's students at Blekinge Institute of Technology. Many of the students have extensive experience from software development. As part of their Bachelor degree, they have obtained practical training in software

	development. Several of the Master students also work in industry in parallel with their studies.
Collection of data and analysis.	Exp. 3: Fault log, time log, Experience questionnaire, Inspection questionnaire, The experiment data are analyzed with descriptive analysis and statistical tests. The collected data were checked for normal distribution. Mann-Whitney test, Chi-square test, Kruskal Wallis.

Article ID	**A14**
Title	Investigating the Role of Use Cases in the Construction of Class Diagrams
Author	Bente Anda and Dag I.K. Sjøberg
Year	2005
Source	Empirical Software Engineering. 10, 3 (Jul. 2005), 285-309.
Type of study	Experiment
Intent	Investigate empirically advantages and disadvantages of two alternative ways (derivation technique and validation technique) of applying a Use Case model in an object-oriented design process. Compare the use of pen and paper with the use of a commercially available modeling tool regarding the two techniques.
Hypotheses	H10: There is no difference in the completeness of the class diagrams. H20: There is no difference in the structure of the class diagrams. H30: There is no difference in the time spent constructing the class diagrams. (Both experiments).
Result	The validation technique resulted in class diagrams that implemented more of the requirements. The derivation technique resulted in class diagrams with a significantly better structure than did the validation technique in the student experiment and slightly better structure in the experiment with the professionals. There was no difference in time spent between the two techniques. In the experiments the use of the tool did have an effect on the results. Based on these results, it may be beneficial to derive classes directly from the Use Cases when the Use Case model contains many details and there is a strong need for good structure, but that otherwise it is better to apply the Use Case model in validation.
Future work	- Conduct further studies to investigate how to apply a Use Case model in an object-oriented design process. - Increase the size and complexity of the task, use different Use Case format - Compare different tools by having a larger number of subjects using each tool, 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 of the quality of the resulting class diagrams by combining several aspects.
Development phase	Transition from functional requirements to object-oriented design.
Application domain	Experiment domain: Library system.

Study design	2 controlled experiments. Replicated. Based on pilot experiment. Experiment 2 is a differentiated replication of Experiment 1. Randomized block experimental design. Students were divided into two groups, one group with pen and paper one with UML-tools. Professionals used their familiar development tool.
Location	Oslo
Exp. year	
Duration	2,5 to 4,5 hours.
Case tools	Experiment 1: Tau UML Suite, pen and paper. Experiment 2: Visio, Rational Rose, Magic Draw.
Software artefacts	Textual requirements document and Use Case model.
Participant details	**Experiment 1:** 53 Subjects were students (third or fourth year of study) taking an undergraduate course in software engineering. They had learned the basics of object-oriented programming and UML through this and one previous course. The subjects were paid to participate in the experiment. **Experiment 2:** The subjects were 22 consultants from eight companies. All of them had used UML on software development projects. More than half of them had studied UML as part of their education. The companies were paid to participate in the experiment.
Collection of data and analysis.	The subjects uploaded documents produced through a web based tool for experiment support. Feedback collection tool (only experiment 1). Analyze the class diagrams and rating them by Author and extern.

Article ID	**A15**
Title	OPM vs. UML - Experimenting with Comprehension and Construction of Web Application Models
Author	Iris Reinhartz-Berger, Dov Dori.
Year	2005
Source	Empirical Software Engineering, Volume 10, Issue 1, Jan 2005, Pages 57 - 80
Type of study	Experiment
Intent	Evaluating comprehension and construction quality of OPM in comparison to UML in the domain of web applictions. Conallen's extension to UML is compared to OPM.
Hypotheses	1. Questions which can be answered by inspecting a single UML view would be more correctly answered when UML rather than OPM is used. 2. OPM will be more adequate than UML for understanding the dynamic aspects of a system and the complex relations among various (structural and dynamic) system modules. 3. OPM was expected to be more correctly and more easily applied than UML for modelling complex, dynamic applications.
Result	8 out of 9 questions scored higher when the system was modelled using OPM than when it was modelled using UML. In particular, the construction problems for both systems scored higher when students were required to use OPM. In both case studies, when using OPM the students answered the distribution questions more correctly, but these differences were not statistically significant. For the structure comprehension category questions, the students' results were significantly better when using OPM in the project management system, while in the book ordering application the students' results in this category were insignificantly better when they used UML. Comparing OPM to UML in terms of Web applications comprehension and modelling quality, we concluded that the single OPM diagram type, the Object-Process Diagram (OPD), which supports the various structural and dynamic aspects throughout the system lifecycle, is easier to understand and apply by untrained users. The results suggest that OPM is better than UML in modelling the dynamics aspect of the Web applications. In specifying structure and distribution aspects, there were no significant differences. The results further suggest that the quality of the OPM models students built in the construction part were superior to that of the corresponding UML models.
Future work	Future work should validate our findings with analysis and design experts who are familiar with these languages. Further experiments should also be carried out to compare OPM to other leading modelling languages and methods.

Development phase	Design.
Application domain	Web applications. Experimental tasks: project management system and a book ordering application.
Study design	Two groups, two test form types. The two case studies were designed to be identical in their scope, or size. Final three-hour examination of the course. The questions on both models for the same case study were identical.
Location	Technion, Israel Institute of Technology
Exp year	Spring semester 2002
Duration	3 hours.
Case tools	
Software artefacts	UML (deployment, Statecharts, site map, class, sequence) and OPM-diagrams/models.
Participant details	81 Third year students in a four-year engineering B.Sc. program at the Technion, Israel Institute of Technology, who took the course ‘‘Specification and Analysis of Information Systems’’. They had no previous knowledge or experience in system modeling and specification. During the 13-week course the students studied three representative modeling notations: DFD for two weeks, UML for five weeks, and OPM for two weeks. They then also studied how to model Web applications in UML and OPM for one additional week each. The students were required to submit four modeling assignments in order to practice the use of DFD, UML (Use Case, Class, and Sequence Diagrams), Statecharts, and OPM.
Collection of data and analysis.	All the questions about the project management system (in both UML and OPM) were graded by one of the two teaching assistants, while the questions about the book ordering application were graded by the other.

IEEE Transactions on Software Engineering:

Article ID	**A16**
Title	An Experimental Comparison of Usage-Based and Checklist-Based Reading
Author	Thomas Thelin, Per Runeson, and Claes Wohlin
Year	2003
Source	IEEE Transactions on Software Engineering vol. 9 No. 8 August 2003
Type of study	Experiment.
Intent	Compare usage-based and checklist-based reading. Reading techniques, Inspection.
Hypotheses	H0 Eff — There is no difference in efficiency (i.e., found faults per hour) between the reviewers applying prioritized Use Cases and the reviewers using a checklist. HA Eff — There is a difference in efficiency between the reviewers applying prioritized Use Cases and the reviewers using a checklist. H0 Rate — There is no difference in effectiveness (i.e., rate of faults found) between the reviewers applying prioritized Use Cases and the reviewers using a checklist. HA Rate — There is a difference in effectiveness between the reviewers applying prioritized Use Cases and the reviewers using a checklist. H0 Fault—The reviewers applying prioritized Use Cases do not detect different faults than the reviewers using a checklist. HA Fault—The reviewers applying prioritized Use Cases detect different faults than the reviewers using a checklist.
Result	The main results from the analysis are that reviewers using UBR find more critical faults and do it more efficiently. Efficiency—Reviewers using usage-based reading are significantly more efficient than reviewers using checklist-based reading. This difference is significant for all faults and for critical faults. Effectiveness—Reviewers using usage-based reading are significantly more effective than reviewers using checklist-based reading. This difference is significant for critical faults, but not for all faults. Faults—Reviewers using usage-based reading find different and more unique faults and, especially, more critical faults than reviewers using checklistbased reading. Teams—The team analysis also shows that usage based reading is more effective and efficient than CBR. This is true for all team sizes ranging from two to six reviewers. Fault Finding—A reviewer applying usage-based reading starts to find faults earlier than a reviewer using CBR. The differences for all faults are about 20 minutes and this difference is even larger for critical faults.

Future Work	- Enhance UBR, either to include checklist items or to investigate the time-based ranking method. - Replicate and compare with, for example, usage-based testing. - Replicate in different context and address changes in the design, for example, use a different domain, and seed more faults into the document under inspection. - Investigate the method in a case study in an industrial setting in order to evaluate whether it still provides positive effects. Investigate the method with professionals as subjects. - Investigate UBR with time-controlled reading. - Investigate a hybrid of UBR and CBR.
Development phase	Inspection, which is performed in all phases. Inspection of a design document.
Application domain	Taxi management system
Study design	Controlled. Two groups. Blocking on experience. The Use Case document contains 24 Use Cases. The design document contains 38 faults, of which two are new faults found during the experiment and eight are seeded faults injected by the person who developed the system. The 28 others are faults made during development of the design document and later found in inspection or test. These faults were reinserted prior to the experiment.
Location	
Exp year	Spring 2001.
Duration	2 days.
Case tools	
Software artefacts	One requirements document (natural language), one design document written in the specification and description language (SDL)(includes two MSC.), one Use Case document, and one checklist.
Participant details	23 Fourth-year software engineering master's students at Blekinge Institute of Technology in Sweden. Many of the students have extensive experience from software development. As part of their bachelor's degree, they have obtained extensive practical training in software development. The experiment was a mandatory part of a course in verification and validation.
Collection of data and analysis	A questionnaire was used to explore the students' experiences. Descriptive analysis and statistical tests. Checked as to whether they follow a normal distribution. Since no such distribution could be demonstrated using normal probability plots and residual analysis, nonparametric tests are used. The Mann-Whitney test is used to investigate hypotheses HEff and HRate and a chisquare test is used to test HFault.

Article ID	**A17**
Title	An Experimental Investigation of Formality in UML-Based Development
Author	Lionel C. Briand, Yvan Labiche, Member, Massimiliano Di Penta, and Han (Daphne) Yan-Bondoc
Year	2005
Source	IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 10, OCTOBER 2005
Type of study	Experiment.
Intent	Evaluate whether the OCL has an impact on three software engineering activities, that is, 1) the detection of defects in UML models, 2) the comprehension of a system's functionality, behavior, and structure based on UML models, and 3) the maintenance of UML documents, with a particular focus on change impact analysis.
Hypotheses	H0: There is no difference in the subjects' Comprehension (C) Maintenance (M), and Defect Detection (D) effectiveness while working on UML analysis documents using or not using OCL. Ha: using OCL improves effectiveness for all three dependent variables.
Result	The results show that, once past an initial learning curve, significant benefits can be obtained by using OCL in combination with UML analysis diagrams to form a precise UML analysis model. However, the benefits for each task are modest and become practically significant only when taken all together. But, this result is however conditioned on providing substantial, thorough training to the experiment participants. Furthermore, such benefits are strongly dependent on the ability, experience, and training of software engineers.
Future Work	- Investigate if the benefits of devising OCL expressions justify their cost. - Take into account how to measure cost, the quality of implementation, and the correctness and completeness of UML models. - Re-evaluate the cost-benefit of defining precise UML models with OCL constraints when tools are available.
Development phase	Analysis and design.
Application domain	Experimental tasks: Cab Distribution (CD) system and a Video Store (VS) system.
Study design	Compulsory laboratory exercises. Controlled. 4 experimental groups. Each group existed of 9-10 students (a total of 38) This paper reports on a series of two controlled experiments. 4 labs, each lab being a week apart. Two blocks (High Ability and Low Ability). Grouped according to

	whether they learned OCL in a prerequisite course or in the course (the material and number of hours of lectures used were the same), and according to the undergraduate engineering program they were registered in. Each of the four student groups was then randomly assigned subjects from blocks in nearly identical proportions.
Location	Carleton University, Ottawa, Canada.
Exp. year	
Duration	Exp 1: 4 weeks, Exp 2: 8 weeks
Case tools	
Software Artefacts	Use Case diagram along with Use Case descriptions, sequence diagrams, Statecharts, textual description of their states and transitions, a class diagram, and a data dictionary
Participant Details	Exp. 1: 38, Exp. 2: 84 The context of the experiment is a fourth year Computer and Software Engineering course. The last, most advanced software engineering course in their four-year Bachelor program. The students have all been trained in UML-based object-oriented software development in at least three previous courses, with an increasing focus on software modelling. Students were not graded on performance but were expected to perform their tasks individually in a professional manner to obtain the points assigned to the laboratory. They were aware of the pedagogical purpose of the exercises—that is to experience modelling tasks in the presence or absence of OCL—but did not know the exact hypotheses tested. More training was administered before the second experiment trial than in the first trial.
Collection of data and analysis	UML analysis documents with or without OCL constraints, and with or without seeded defects, questionnaires for the Comprehension and Maintenance tasks, and a post lab survey questionnaire. Additionally, to verify if the blocks are appropriate, a pre-lab survey questionnaire is administered to obtain information about the background of the subjects. Defect Detection task: The subjects' performance for this task is measured as the percentage of seeded defects detected. Comprehension task: The subjects' performance for this task is measured as the percentage of correctly answered comprehension questions. Maintenance task: The subjects' performance for this task is measured as the percentage of affected model elements correctly identified. A complementary measure that was considered but not presented here due to space constraints is the total number of wrongly identified model elements: We did not observe significant differences.

The Requirements Engineering Journal:

ArticleID	**A18**
Title	Deriving Goals from a Use-Case Based Requirements Specification
Author	Annie I. Anto´n, Ryan A. Carter, Aldo Dagnino, John H. Dempster, Devon F. Siege
Year	2001
Source	Requirements Engineering (2001) 6: 63-73
Type of study	Case study
Intent	- Employ the goal and scenario identification and elaboration heuristics available in the GBRAM (Goal-Based Requirements Analysis Method). - Report on experiences in managing a large collection of Use Cases during the requirements specification activities for an electronic commerce application
Results	- Lack of contextual information increases the risk that system requirements may be misinterpreted. We provided this context by always attaching a goal to each Use Case. - It is valuable to separate user goals from system goals by expressing achievement goals accordingly. Achievement goal categories lead to the derivation of a more complete set of goals and viewpoint analysis. - Missing and inconsistent naming of Use Cases is indicative of an incomplete and flawed specification - The list of included and extended Use Cases often pointed to undefined or non-existent Use Cases. Additionally, some referenced Use Cases were never defined. To identify missing Use Cases, we created an 'includes tree' to track relationships between Use Cases and discovered 15 missing Use Cases. - Due to the lack of requirements management tool support, a significant amount of overhead was incurred. Whereas maintaining pre-traceability requires dutiful attention, it can be greatly simplified with appropriate tool support. Introducing additional traceability can affect cost and scheduling estimates. However, the gains and benefits well outweigh the costs [26]. - Domain-specific goal classes help ensure better requirements coverage - A Use Case collection is not a suitable substitute for an SRS. - Our study gives evidence of software practitioners adopting a Use Case collection as a suitable substitute for a requirements specification.
Development phase	Requirements engineering.
Domain studied	Electronic commerce.
Future work	- Further research is needed and advances in the role of Use Cases and

	goals during requirements specification as well as the need to investigate and develop more appropriate evolutionary requirements and software process models for rapid development environments. -Further extend and refine the GBRAM. - Continue the collaboration with project sponsor to focus on defining appropriate evolutionary prototyping models that provide analysts with feedback and additional insights into the evolving set of requirements needed to build a given system. - Further investigation for the evidence of software practitioners adopting a Use Case collection as a suitable substitute for a requirements specification.
Location	
Duration	Sessions ranging from one to three hours in duration, once a week for two months.
Case tools	
Software artefacts	Use Case model, requirements pattern.
Participant details	
Collection of data and analysis	Individual analysts performed goal analysis of agreed-upon Use Cases that were then discussed and revised while collaboratively recording all goals and auxiliary notes. During these sessions they applied the GBRAM. The GBRAM was employed to analyse an existing requirements specification. Goals and their associated information were identified, numbered, classified and stored. The information was tracked. Goals were named using meaningful keywords selected from a predefined set of goal categories.

Article ID	**A19**
Title	Are Use Case and class diagrams complementary in requirements analysis? An experimental study on Use Case and class diagrams in UML
Author	Keng Siau, Lihyunn Lee
Year	2004
Source	Requirements Engineering (2004) 9: 229-237
Type of study	Experiment
Intent	Compare Use Case and class diagrams. Investigate whether these two diagrams are able to complement each other in the context of understanding system requirements.
Hypotheses	– H1: The completeness of interpreting class diagrams and Use Case diagrams is different. – H2a: The inclusion of Use Case diagrams affects the completeness of the problem domain interpretation using class diagrams. – H2b: The inclusion of class diagrams affects the completeness of the problem domain interpretation using Use Case diagrams. – H3: The sequence combination of the diagrams affects the completeness of the problem domain interpretation. – H4: Perceived Usefulness is different between Use Case diagrams and class diagrams. – H5: Perceived Ease of Use is different between Use Case diagrams and class diagrams.
Result	Hypothesis H2a and H2b were rejected. - The results show that the Use Case diagrams were more completely interpreted than the class diagrams. - The presence or absence of one diagram when interpreting another diagram had no effect on the outcome of the interpretation. - There is no statistical difference between class diagrams and the Use Case diagrams for perceived usefulness. - There is no significant difference in the perceived ease of use between class diagrams and Use Case diagrams.
Development phase	Requirements analysis.
Application domain	ATM banking system and music club.
Future work	- Replicate this study using different research methodologies or other subjects. - Investigate the rest of the modeling diagrams in UML, such as activity diagram, sequence diagram, and statechart diagram. - Determine the core UML diagrams and the core constructs in each diagram. - Assert a need for the coexistence of class diagrams and Use Case diagrams for effective requirements analysis.

Study design	Randomly assigned to one of the two treatment groups
Case tools	
Software artefact	Use Case and class diagram.
Participant detail	31 university student volunteers who had completed at least one object-oriented UML course.
Collection of data and analysis	Experiments capturing subjects' performance via questionnaires and process-tracing method were carried out. The final data was obtained from counts of matching information elements identified by the subjects during protocol analysis. A scatter plot diagram and a histogram. Chebyskev's Rule. Verbal protocol analysis. Audiotaped and later transcribed, coded, and analyzed.

Article ID	**A20**
Title	Scenario advisor tool for requirements engineering
Author	Jae Eun Shin, Alistair G. Sutcliffe, Andreas Gregoriades
Year	2005
Source	Requirements Eng (2005) 10: 132–145
Type	Experiment
Intent	Investigate the usefulness of a scenario advisor tool, which was built to help requirements engineers to generate sufficient sets of scenarios in the domain of socio-technical systems.
Hypotheses	
Result	The overall user performance for writing new scenarios was significantly better with the tool. For writing variations of scenario provided, the overall user performance was also significantly higher with the tool. The subjects used more information, and scenario components with the tool than with the paper-based method, and this may have led to better task performance. The scenario advisor tool helped users to write sounder scenarios without any domain knowledge, and is also useful for generating more variations of existing scenarios by providing scenario generation hints for each property of the model components.
Development phase	Requirements engineering.
Application domain	Socio-technical systems. Experiment applied in the Military domain.
Future work	Refine the scenario advisor tool. Develop a help system for a step-by-step scenario generation procedure or a scenario template and providing domain-specific information with more examples. Develop an automatic scenario-generation tool.
Duration	
Case tools	Scenario advisor tool.
Software artefacts	Scenarios. Paper-based information (scenario taxonomy tables and schema diagrams)
Participant details	8 postgraduates and 2 researchers (mean age=28 years; mean computer use=10.3 years; 6 male/ 4 female) participated in the experiment without the tool. None had experience in scenario-based design, although three had some experience in writing scenarios at a novice level. Only one participant was familiar with the military domain. 9 postgraduates and 1 researcher (mean age=27.5 years; mean computer use=10.5 years; 6 male/ 4 female) participated in the experiment with the tool. None had experience in scenario-based design although three had some experience in writing scenarios at a novice level. Two participants said they were familiar with the military

	domain, but they were not domain experts.
Collection of data and analysis	Performance data was assessed by comparing subject scenarios to a standard solution. For each sub-task, we graded the scenario produced on a 50 point scale. Pre-test questionnaires were used to collect user profiles, while post-test questionnaires assessed user satisfaction. We used observation notes and audio recordings of evaluation sessions to analyze usability problems and users' scenario-generation strategies. Debriefing interviews followed up observed usability problems and collected user suggestions for improvements.

Article ID	**A21**
Title	Testing the predictive ability of a requirements pattern language
Author	Peter Merrick, Patrick Barrow
Year	2005
Source	Requirements Engineering (2005) 10: 85–94
Type of study	Case study
Intent	Evaluate the predictive ability of a requirements pattern. Investigate whether an accurate Use Case representation can be constructed from a loosely defined customer requirements statement. A comparison of functional requirements before and after the system is built.
Result	Only one Use Case predicted from the initial diagrams was not implemented in the final system The final system contained more Use Cases than those predicted by the output from the pattern language. In the final system, there were 30 Use Cases delivered, of which 21 were predicted, or 70%. Of all identified Use Cases, fully 23% were assigned to the wrong actor in the first modelling iteration. The application of the RPL is an effective procedure for generating a model quickly, at an early stage in the process, from a non-structured requirements statement.
Future work	- Improve pattern, but how and where will not become clear until they have been used more often. - Assemble a body of evidence where the RPL had been applied. - Study the role of Use Cases in wider project management, such as the reporting of progress, and the management of requirements that change. Explore the relationship between the ability to predict the model of a finished application and the estimation of effort. This work will follow on from that explored in applications of the Use Case Points Method.
Development phase	Procurement phase or ‘‘pre-analysis’’; the informal software engineering lifecycle phase
Application domain	A Web-enabled database with end user and call centre operator interfaces. Website and database.
Location	Norfolk, England.
Duration	
Case tools	
Software artefact	Use Cases and Requirements pattern.
Participant details	The 2 authors of this paper have applied the method. It is probably the same two that developed the requirements pattern.
Collection of data and analysis	The original Use Case model was created through the application of a requirements pattern language designed to be employed during the procurement phase of an IT system. The final Use Case model was

	reverse engineered from the working application. Comparison of whether Use Cases defined in the first iteration remain in the second iteration. The second comparison looks at the models from the perspective of the built system, and compares the number of new user-goal Use Cases that were not predicted in the initial model.