Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance

Wojciech James Dzidek

Thesis submitted for the degree of Ph.D.

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

July 2008

© Wojciech James Dzidek, 2008

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
Nr. 796

ISSN 1501-7710

All rights reserved. No part of this publication may be
reproduced or transmitted, in any form or by any means, without permission.

Cover: Inger Sandved Anfinsen.
Printed in Norway: AiT e-dit AS, Oslo, 2008.

Produced in co-operation with Unipub AS.
The thesis is produced by Unipub AS merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

Unipub AS is owned by
The University Foundation for Student Life (SiO)

Abstract

The Unified Modeling Language (UML) is the de facto standard for object-oriented software analysis and design modeling. The assessment of costs, risks, and benefits of employing UML during software development is often based on subjective opinion rather than scientific evidence as scant and inconsistent evidence is available. This thesis explores the impact of UML with a focus on the maintenance of object-oriented software. It reports on both a comprehensive, systematic review and a large scale, realistic experiment on this topic.

The review looks at the most effective ways of using UML, the experiences of working with commercial UML tools, the associated learning curve, how UML being used in the industry, and the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry. One of the findings is that few empirical studies exist that investigate the costs and evaluate the benefits of using UML in realistic contexts. Such studies are needed so that the software industry can make informed decisions regarding the extent to which they should adopt UML in their development practices.

This thesis also presents the first controlled experiment that investigates the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real, non-trivial system, using professional developers as subjects, working with a state-of-the-art UML tool during an extended period of time. The subjects in the control group had no UML documentation. Results show that the subjects in the UML group had on average a practically and statistically significant 54% increase in the functional correctness of changes ($p=0.03$), an insignificant 7% overall improvement in design quality ($p=0.22$)—though a much larger improvement was observed on the first change task (56%)— at the expense of an insignificant 14% increase in development time caused by the overhead of updating the UML documentation ($p=0.35$).

The thesis concludes with a synthesis of the results from both the review and the experiment, with the outcome suggesting that the benefits of using UML do outweigh the costs and risks in the context of object-oriented software maintenance.

Acknowledgements

First and foremost, I would like to thank my supervisors Erik Arisholm and Lionel Briand for their guidance, input, support, and patience – this thesis would not be possible without them.

Furthermore, I would like to thank:

- The people in the Software Engineering group, I will especially miss the lunchtime banter.
- Dag Sjøberg for his continued support throughout the years.
- Hans Christian Benestad and Marek Vokac whom shared their knowledge and wisdom with me on many topics.
- Magne Jørgensen for the invaluable opinions and the collaboration on the BESTweb experiment.
- Kjetil Moløkken-Østvold, Stein Grimstad, Jo Hannay, Vigdis By Kampenes, and Tore Dybå for the insightful scientific discussions.
- Gunnar Bergersen for being a supportive and inspirational office mate.
- Bjørn Fredrik Nielsen for the fascinating discussions.
- Tanja Gruscke for her contributions to the BESTweb experiment.
- The Simula management and administration, especially Aslak Tveito.
- The employees at Simula, for making the environment pleasant, interesting, and inspiring.

Finally, thanks to my friends and family for supporting me throughout these years.

Table of Contents

1 Introduction

1.1 Problem Definition

UML allows for the visual representation of a system's specification at various levels of design and is used to construct and document an object-oriented software system. This in turn aids in the communicating and understanding of various system properties with various stakeholders. Advocates of UML often cite the following advantages: ability to handle the growing complexity of software development by working at higher levels of abstraction, traceability from requirements to low-level design, and more efficient communication. In fact, engineering designs are traditionally conveyed via two types of complementary notations: textual and visual domain-specific, standardized notations. Since software designs must be expressed and communicated to many stakeholders, a visual language that is complementary to the code should be able to provide advantages as it does in other disciplines. Further, the need to effectively communicate design intent during development, maintenance, and evolution is an area in which improvements can have significant benefits.

Despite growing popularity, there is little reported evaluation of the use of UML-based development, and many still perceive the development and maintenance of analysis and design models in UML to be ineffective [1]. Such practices are therefore viewed as difficult to apply in development projects where resources and schedules are tight. It is then important, if not crucial, to investigate whether the use of UML can make a practically significant difference that would justify the costs. This is particularly true in the context of software maintenance which consumes most of software development resources: "Maintenance typically consumes 40 to 80 percent of software costs. Therefore, it is probably the most important life cycle phase of software." [2], and "60 percent of software's dollar is spent on maintenance, and 60 percent of that maintenance is enhancement. Enhancing old software is, therefore, a big deal." [3]. Furthermore, the maintenance tasks are not necessarily performed by the original developers, in which case a lot of effort must be spent on understanding its functionality, architecture, and a myriad of design details of the large and complex existing system in order to change it correctly.

Having established the need for empirical studies where developers use UML during the maintenance phase leads us to the next issue: the manner in which UML should be used, e.g., the amount of detail that should be present in the diagrams and the necessary tool

support. At one extreme, some argue for using UML at a very informal level where diagrams are sketched on a whiteboard in order to help communicate ideas and alternatives with colleagues; their emphasis is on selective communication rather than complete specification. These diagrams are either soon discarded or quickly become inaccurate (since they don't get modified along with the code). At the other extreme, proponents of Model Driven Architecture (MDA) believe that future programmers should mostly deal with models instead of focusing on code, that is, UML becomes the "programming" language [4]. Since all changes occur via the models, these are always up-to-date, though the opposition claims that this is highly inefficient. This approach depends on tools that we do not yet possess though an increasing number of sophisticated modeling tools are available.

1.2 Goal

The goal of this thesis is to investigate the costs and benefits of UML during software maintenance and evolution in the context of object-oriented (OO) software by (1) aggregating and synthesizing existing evidence in an extensive yet systematic manner and (2) directly extending the body of evidence via a large and realistic controlled experiment.

1.3 Contribution

This thesis contains two main contributions regarding the effects of UML used by developers during the maintenance of OO software. Each of the two contributions can be subdivided into a methodological contribution and the final results. The methodological contribution should not be underestimated as there are no clear guidelines available for performing the type of studies presented in this thesis.

First, an extensive systematic review was performed on the topic to aggregate and synthesize published knowledge. Thirteen top journals and conferences were selected, of these 1572 papers were inspected for relevance, yielding 23 relevant papers. Information from the relevant papers was then synthesized and presented in a structured manner. The advantages of systematism over an ad-hoc method are: thoroughness, fairness – no relevant papers can be omitted, and repeatability – can be used to gauge how knowledge on the topic evolves. The methodological contribution of this study is a method for performing systematic reviews on topics where query-based approaches cannot be used.

The review was used to address the following six research questions as well as to identify holes in the existing research and guide future research efforts:

- What are the costs, risks, and benefits of using UML?
- What are the most effective ways of using UML?
- What are the experiences of working with commercial UML tools?
- Learning curve: how hard is it to learn UML in practice?
- How is UML being used in the industry?
- What are the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry?

The main results, apart from the surprisingly few papers identified on the topic, suggest that the benefits of UML outweigh the costs and risks; employment of constraints is underused considering the benefits that they bring; and a very important area receives very little attention – diagram use, investigating the type of information, amount of information, and the way in which the diagrams should be used together in order to maximize efficiency.

The second main contribution was a large and realistic controlled experiment that looked at the costs and benefits associated with UML during maintenance and evolution. This experiment complements the systematic review as it addresses a hole in the existing research. It is the first experiment on UML to have such a high level of realism: it involved 20 professional developers (intermediate to senior level consultants) individually performing the same five maintenance tasks to the same real, non-trivial system where ten of the developers worked with a UML-supported development environment and UML documentation, whereas the other ten developers used the same tools but had no UML documentation to read or update. The developers took one to two weeks to implement the change-tasks.

The experiment addressed the following four research questions:

- Does the provision of UML documentation reduce the effort required in correctly implementing the change tasks?
- Does the provision of UML documentation increase the functional correctness of the delivered solution? Since a fault found after the release of the software is significantly more expensive to fix than one found during development [5, 6], special attention must

be paid to whether UML increases the probability of the change being functionally correct.

- Does the provision of UML documentation improve the design quality of the delivered solution? Alternatively, does the use of UML decrease the decay of a system's design caused by maintenance tasks?
- What are the shortcomings of the used state-of-the-art UML tool and how can it be improved?

The methodological contribution consists of showing how to design, conduct, and analyze such a large and realistic experiment. Note that the appendices shed more light on the amount of detail that went into the experiment materials to ensure realism.

The main results show that UML allowed for traceability between functional requirements and code, significantly increased functional correctness of changes by 54%, improved design quality by 7% – though this was overall statistically insignificant and a much larger and significant improvement was observed on the first change task (56%), at the expense of an insignificant 14% increase in development time caused by the overhead of updating the UML documentation.

This thesis is relevant to two groups of people: practitioner and researchers. Practitioner can use the results in this thesis to learn the state of the art in terms of what is possible, attainable, most effective, and problematic. In turn, such information can be used to decide about whether and how to introduce UML in a development organization. Researchers can learn from the presented experiences of having conducted such a large and complex experiment and from the identified future research areas.

Note that a part of this work (the controlled experiment) has already been published in IEEE Transaction on Software Engineering [7].

The dvision of labour of the work presented in this thesis is summarized in Table 1.1.

Table 1.1: Division of Labour

Year	Phase	Role
2004 Jan – 2005 Dec	Design of the BESTweb experiment.	Primary, in collaboration with Magne Jørgensen and my supervisors.
2004 December	Pilot experiment	Primary role.
2004 Feb – 2004 June	Implementation of BESTweb system from specifications.	Primary role; the specifications for the system were provided by Magne Jørgensen.
2004 Jan – 2005 Feb	Preparation of the experiment materials.	Primary role.
2004 Nov – 2005 Aug	Recruitment of subjects.	Hans Christian Benestad
2004 Nov – 2005 Aug	Execution of the experiment.	Primary role.
2005 Sept – 2007 May	Analysis/write-up of the experiment.	Primary role.
2007 June	The BESTweb experiment is submitted for publication.	N/A
2007 Jan – 2008 July	The systematic review is conducted and written up.	Primary role.
2008 November	The systematic review paper is submitted for publication.	N/A

1.4 Thesis Organization

This thesis is organized in eight chapters. In addition, there are five appendices providing details on the experiment materials and the experiment data. Table 1.2 gives an overview of the chapters of this thesis.

Table 1.2: Overview of the thesis organization and content

Chapter	Content
Chapter 1	**Introduction**: Introduces the thesis by explaining the problem that is being addressed, the specific goals, the contributions, and the organization of the thesis.
Chapter 2	**Background**: A background on relevant topics – software maintenance, UML, and empirical software engineering – is given. This chapter also explains why each of these topics is important.
Chapter 3	**Research Objectives and Method**: Having provided the background on the relevant topic in the previous chapter, this chapter positions the research within the wider scope of those topics.
Chapter 4	This chapter presents the systematic review: **A Systematic Review on the Effects of UML during the Maintenance of Object-Oriented Software**
Chapter 5	This chapter presents the experiment: **A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance**
Chapter 6	**Conclusion and Future Work**: In this chapter the results of both the systematic review and the experiment are synthesized, conclusions are drawn, and future work is outlined.
Appendix A	**Multivariate Analysis**: Presents the multivariate analysis and results for the experiment.
Appendix B	**Detailed Qualitative Analysis Results**: Shows a detailed breakdown of the data – per subject, for the experiment.
Appendix C	**The Tasks**: Each of the five task descriptions that were presented to the subjects of the experiment.
Appendix D	**UML Documents**: The UML documentation of the BESTweb system that was made available to the UML group of subjects, in the experiment.
Appendix E	**BESTweb Architecture Document**: The architecture/user's manual of the BESTweb system that each subject received.

2 Background

The main topic of this thesis is to investigate, from an empirical perspective, the costs and benefits of UML in software maintenance. Before such a discussion can be undertaken it is imperative to define the concepts under study in a precise manner to ensure a common understanding of the individual concepts in the discussion and how they interrelate. The goal of this chapter is to accomplish this with respect to Software Maintenance, UML, and Empirical Software Engineering.

2.1 Software Maintenance

This thesis focuses on the maintenance phase of software development. Software maintenance is defined in ISO/IEC 12207 [8] as the process that occurs when software undergoes modifications to code and associated documentation due to an error or the need for improvement or adaptation. The maintenance phase is arguably the most important one as it the largest and costliest, as discussed in [2] and [3]: “Maintenance typically consumes 40 to 80 percent of software costs. Therefore, it is probably the most important life cycle phase of software.”, and “60 percent of software's dollar is spent on maintenance, and 60 percent of that maintenance is enhancement. Enhancing old software is, therefore, a big deal.”

A more fine-grained definition of maintenance is given by Rajlich and Bennett in [9] where they divide the software lifecycle into five distinct stages: initial development – the development of the first functioning version, evolution – work on extending the system's capabilities to meet user needs, servicing – fixing minor defects and implementing minor functional changes, phase-out – the servicing of the software is terminated, and closedown – the software is withdrawn from the market. Given that categorization, this thesis is interested in the evolution and servicing phases of the software lifecycle.

Software maintenance is a difficult activity worsened by the fact that often developers performing the work are inexperienced with the system. A developer must understand the existing system before they perform changes on it. Without proper understanding the developer may break existing functionality and deteriorate the architecture, accelerating code decay [10]. Software complexity impedes efforts to understand the existing system and makes it more difficult to implement changes.

Maintainability is a closely related concept that is difficult to define precisely and even more difficult to quantify. Maintainability is loosely defined as the ease with which a software system can be modified to correct a fault or conform to change requirements. Considering how important and costly the software maintenance phase is, it is important to identify factors that increase maintainability (to maximize the return on investment) and retarded code decay.

Documentation can increases maintainability [11, 12] by capturing rationale, explaining the architecture of the system (at a higher level of abstraction than the code), and providing traceability between the functional requirements and the code. This comes at a cost of having to keep it up-to-date.

Early decisions made during the planning of the software system, apart from documentation, can ensure that a development paradigm that eases future system maintainability is selected, like OO instead of procedural [13]. Further, the architecture can be designed in a way that makes the system more resilient to change, by, for example, employing design patterns [14]. Finally, during the development of the system heuristics like low coupling/high cohesion [15] and adherence to coding standards may further help increase system maintainability.

Before performing a maintenance task an impact analysis should be performed to understand the ramifications of the change. During the implementation of the change reverse engineering tools can help understand the existing system. Dynamically enforced contracts [16] and assertions help understand the system while decreasing the odds of break existing functionality. After the performing the change, the presence of a regression test suit further helps ensure that the change did not break existing functionality.

Ensuring software maintainability is a very hard problem compounded by the fact that it is very difficult to measure – there is no such metric as a *maintainability factor* – and is often performed by junior developers or developers simply unfamiliar with the system. This is exemplified in an experiment evaluating the claim that a delegated control style of OO programming increases the software's maintainability (versus a centralized control style) [17]. The claim is found to be true for senior developers, but not for undergraduate students and junior developers.

2.2 UML

This chapter briefly introduces UML [11] by first giving a general background on the subject, followed by an overview of the diagram types available and the available extensibility mechanisms. Next, different degrees of formality at which UML can be used are discussed. Having introduced the core UML constructs, a discussion on how UML is used in the different software activities follows. Lastly, a short discussion on the potential costs and benefits of adopting UML is given. Note that this thesis focuses on UML 1.4 [18] though references to UML 2.0 [19] are made when appropriate.

2.2.1 What is UML?

UML is the de facto standard for modeling object-oriented software. The UML notation is a fusion of three preceding methods: OOAD [20], OMT [21], and OOSE [22]. UML enables developers to specify, visualize, and document software systems via models. Models are abstractions of reality focusing on showing details important to understanding a particular facet while hiding irrelevant detail. UML is a general-purpose modeling language that all modelers can use, not a complete development method – it is intended to support existing development processes.

In this thesis the focus is on the software maintenance phase and ways in which software maintainability can be increased (Section 2.1). UML is said to increase maintainability in terms of more efficient form of communication for OO systems. This is especially useful when non-original developers must understand a system they are unfamiliar with. Hence, UML could help them get up to speed faster. UML is also said to help deal with complexity as dealing with abstraction may makes it easier to understand large systems.

2.2.2 Types of Diagrams

UML has different diagram types as different diagrams are necessary to reveal unique yet overlapping views of a system. UML has two categories of diagrams, structural and behavioral. Structural diagrams illustrate the static features of a model like classes and association, objects and links, and collaborations. Behavioral diagrams, on the other hand, show the dynamic aspect of collaborations.

Structural diagrams consist of the class diagram, the object diagram, and the deployment diagram. Class diagrams visualize the various kinds of static relationships that exist between object types. Object diagrams, or instance diagrams, show a snapshot of the

objects in a system at a point in time. Deployment diagram show a system's physical layout in terms of which piece of software runs on what piece of hardware.

Behavioral diagrams include the use case diagram, the activity diagram, the interaction diagrams (sequence diagram and collaboration diagram), and the statechart diagram. Use case diagrams capture the functional requirements of a system, they models the users' expectation for using the system. Activity diagrams model workflows in a system. Interaction diagrams provide a bridge between the use case diagram and the class diagram, describing the dynamic view of the system.

Interaction diagrams are used to represent scenarios and show the way in which objects interact, via messages, to perform a task. These diagrams represent the same information (they are isomorphic) but use different notations; the difference is in emphasis. Sequence diagrams emphasize time while collaboration diagrams emphasize object relationships. Further, state diagrams are also used to document the dynamic view of the system. Their emphasis is on modeling systems or parts of system that are composed of finite states and are complementary to interaction diagrams.

2.2.3 Extensibility Mechanisms

The UML extensibility mechanisms – stereotypes, tagged values, and constraints – provide the ability to customize UML diagrams.

Stereotypes are used to create a new model element by introducing new semantics to an existing model element. The new element, targeted for a particular problem domain, has the same structure as an existing element, but with additional constraints, a different interpretation and icon, and different treatment by code generators and other tools.

Tagged values are element metadata used for specifying keyword-value pairs of model elements, where the keywords are attributes. A popular uses of a tagged value is to specify properties that are relevant to code generation tools.

Constraints extend the semantics of a UML construct, enabling the modification of existing rules or the addition of new rules. The Object Constraint Language [23], or OCL, is the formal constraint specification language in UML.

2.2.4 Degrees of Formalism

Modeling can be performed a different degrees of formality, from ad hoc on a simple medium (like paper) [24] to the software being completely specified via models [4], and

everything in between. Various individuals and organizations have defined various degrees of formality, also referred to as levels of modeling, though their visions vary.

The book on OCL and MDA [23] outlines six such degrees in terms of *specification levels*, from 0 to 5: No Specification, Textual, Text with Diagrams, Models with Text, Precise Models, and Models Only. Level 0 is the lowest level where the specification of the software only resides in the heads of the developers. At level 5 the model is a complete, consistent, detailed, and precise description of the system. At this highest level the models are good enough to enable complete code-generation.

Motorola is a strong player in Model-Driven Engineering and has defined six Modeling Challenge Levels (MCE) [25]. The goal is maximum automation via the creation of rigorous models throughout the development process. This is to combat ongoing effort to reduce development costs in spite of increasing system complexity. The six levels are: No Modeling, Informal Modeling, Formal Modeling, Model-Centered Design, Model-Driven Engineering, and Optimized Model-Driven Engineering.

Fowler, in his popular book on UML [24], defines three degrees: UML as Sketch, UML as Blueprint, and UML as Programming Language.

These different scales can be generalized to three degrees of formality (apart from "no modeling"): (1) The lowest level of modeling is ad hoc, informal, on a simple medium (like paper or a whiteboard). (2) Next is formal modeling with tools at a level where the models coexist (are synchronized) with the code. The models are used to show high level design decisions. This level allows for some automation (e.g., simple code generation). Degree (3) is the point at which the entire system is specified using models ("the model is the code"). This level is referred to as Model Driven Architecture (MDA), Model Driven Development (MDD), and Model Driven Engineering (MDE). It allows for the highest degree out automation (e.g., automatic test generation) though it is dependent on highly sophisticated tools that arguably do not yet exist. Degree (3) is subject to a lot of debate and controversy and many believe that MDA will fail [26, 27].

It is important to notice the fact that the higher the degree of formality the more the approach is dependent on tool sophistication. While the largest benefits of modeling come with the high degrees of tool sophistication, many aspects of such tools are still at the research stage [28, 29].

Opinions vary on the optimal degree of formality and as this is an open question: Fowler recommends UML as Sketch as he believes that anything beyond this is "too difficult to do well and slows down a development effort" [24]. The problem with this degree of formality is the fact that such models are either quickly discarded or quickly become out-of-date. Bruegge and Dutoit, authors of a software engineering textbook [12], promote the second degree. This degree helps keep models up-to-date at a cost of modeling tool overhead and the effort of updating the models. OMG strives for the highest degree [4], but this is currently not feasible for general-purpose development. Given these choices, the experiment conducted in this experiment employed the second degree of formality.

2.2.5 Use in the Different Activities

UML serves various purposes during the different activities of software development [11]. During requirements elicitation, use cases and actors are identified. During analysis, developers build a model describing the application domain represented with class and object diagrams and the dynamic model composed of statechart and sequence diagrams. The analysis model focuses on structuring and formalizing requirements and is used to reason about the requirements and help gain new insights and discover errors.

During system design models are refined to account for the design goals and the software architecture. UML helps to visualize the architecture, making it explicit. It also helps to visualize how classes are interconnected and how they interact.

UML also aids testing activities [30, 31]: class and state diagrams aid unit testing; interaction and class diagrams aid functional testing; use case, activity, and interaction diagrams aid system and regression testing.

During software maintenance UML can contribute to both, projects with and without existing UML documentation. In the case where no UML diagrams exists, UML diagrams – e.g., sequence diagrams [32] – can be reverse-engineered from parts of the system of interest. These may then help understanding the existing system. If the UML diagrams do exist, then system understanding may be accelerated thanks to high-level documentation of the system combined with the potential for traceability: relevant system functionality is identified in the use case diagram; from the use case diagram relevant object and methods are identified in interaction diagrams. Classes can be further understood via class and statechart diagrams. Thus, UML diagrams may help during impact analysis and identifying the location in the code where change must be performed.

Note that independent of the activity, whenever the system undergoes change the UML documentation may also have to be updated. Further, after a change it is imperative to ensure that consistency between different related models is sustained. Advanced tools can greatly aid this effort and may even reduce total effort associated with a change if the change can be performed at the model-level.

Last, the fact that change pervades the development process must be recognized and dealt with. This is done via configuration management [33], the process of controlling and monitoring change to work products. UML artifacts are part of the documentation and therefore must be maintained in the configuration management system. This is not trivial as existing configuration management systems are largely geared to supporting textual artifacts, not graphical notations. While this is an important area, it is not covered in this thesis for two reasons: no UML tools supported UML configuration management at the time of experiment design and no papers identified in the systematic review discussed this issue.

2.2.6 Costs and Benefits

The adoption of a nontrivial technology comes with potential costs, benefits, and risks. UML's purported benefits are decreased effort, increased correctness, increased design quality, improved communication, improved documentation, and easier testing [10].

The risk of a new technology is that if it fails to meet the demands it may potentially delay the schedule. The potential costs are that one or more of the aforementioned areas suffer from the opposite effect, e.g., effort increases. One of the reasons why effort may in fact increase is due to having to perform additional tasks such as model construction and maintenance. Communication and documentation could also suffer if, for example, the models mislead the reader.

The framework around the technology must also be in place, this includes: the purchasing of the UML tool, the integration of the tool into the work environment, the training of the staff with UML and the tool, and the adoption of the process to the new technology.

2.3 Empirical Software Engineering

This chapter briefly introduces the field of empirical software engineering [34, 35]. First, background is given on the types of problems addressed by the field. Then, techniques for performing empirical software engineering are outlined and compared.

Empirical studies have traditionally been used in human behavior centered disciplines, like social sciences and psychology where, unlike in physics, the laws of nature do not apply. Software development is strongly affected by human behavior; this is why techniques from social sciences and psychology are so relevant. For example, comparing two sorting algorithms is relatively straightforward, the same cannot be said about comparing the procedural and OO development paradigms.

The problems associated with such evaluations are discussed in [25], an industrial paper, where the authors enumerate through the problems like the lack of a common baseline and the infeasibility of setting up parallel development environments for large-scale development projects. The authors continue that even with the required resources it would be difficult to account for many factors, including experience levels with the software system, experience with the technology, learning curves associated with the technology, and various other experience and human differences.

Thus, empirical software engineering strives to offer objective results that form an important input to the decision-making process regarding adoption of a technology. Unfortunately, empirical software engineering is difficult and expensive. Because of this fact, software techniques are often adopted based largely on advocacy. In fact, even though the need for experimentation in software engineering was emphasized as early as the 1980s [36], it still an area that needs much improvement [37]. If software engineering is to be become a mature and true engineering discipline it must follow the lead of the established engineering disciplines and empirical software engineers is a step in that direction.

There exist three major techniques used to conduct empirical software engineering – surveys, case studies, and experiments. **Surveys** are retrospective studies that investigate relationships and outcomes. Surveys have the ability to provide a large number of variables to evaluate and are especially well-suited for answering questions about what, how much, and how many, as well as questions about how and why [38].

Systematic reviews are surveys based on data from previously published studies for the purpose of research synthesis. The strength of the systematism is that it minimizes the chances of drawing incorrect or misleading conclusions as a result of biases in primary studies or from biases arising from the review process itself [35].

Case studies consist of an in-depth, longitudinal examination of a single instance or event. They are important in understanding the actual practice of software development, and such

understanding is an essential prerequisite to the primary aim of empirical software engineering research – to help guide practitioners [39]. A major strength of the case study approach is that it allows the study of a phenomenon within its real-life context [40]. Case studies are generalizable to theoretical propositions and not to populations or universes [40], in other words, a case study may provide useful insights for practitioners but it cannot provide answers to generic research questions. Case studies are cost-effective for industrial investigation of software engineering methods and tools. Unfortunately, case studies are not ideally suited for evaluating competing technologies; parallel investigations would have to be held and, as earlier discussed, all but very few companies can afford to do this. Controlled experiments address this very problem.

Controlled experiments are employed when control over a situation is desired with direct, precise, and systematic manipulation of the behavior of the phenomenon to be studied [34]. This control can generate stronger statistical confidence in the conclusions [41] and enables the investigators to better understand the issues at stake and the factors to be considered.

The common sentiment towards controlled experiments in software engineering is rather negative: “Unfortunately, since controlled experiments are expensive and difficult to control if the project is too large, the projects studied tend to be small.” [42] – this raises the question about the extent to which the results can be generalized to realistic tasks, artifacts, and project settings (external validity). The experiment presented in this thesis shows that while it is difficult and costly, it is possible and valuable to conduct experiments on nontrivial realistic system with professional developers for an extended amount of time. The benefits of such an undertaking are clear results that are difficult to refute.

All the above described techniques are complementary: surveys are used to gauge the current state or opinions on a topic, systematic reviews help synthesize knowledge in an unbiased manner, case studies help investigate technologies and techniques in industry, and experiments help evaluate and compare different technologies and techniques.

3 Research Objectives and Method

The main research objective of this thesis is to use empirical studies to investigate the effects of UML when used by developers during the maintenance of OO software. This chapter focuses on explaining each part of this objective. First, the OO paradigm is targeted because of its predominance and the fact that it is still in the growth stage [43]. Next, the focus is on the maintenance phase due to this being the largest and costliest part of the software development process (Section 2.1), thus, this is an area where gains might have the largest impact. UML is the technique that is being evaluated, because it is a promising technique of relatively wide-spread use (Section 2.2) and no technique should be adopted without prior evaluation (Section 2.3) – hence this work.

The investigation is conducted by means of two complementary studies; the first is broad in scope while the latter is focused in scope. In the first study, existing knowledge on the topic is synthesized in terms of the costs, risks, and benefits of using UML, the most effective ways of using UML, the experiences of working with commercial UML tools, the learning curve, how UML is being used in industry, and the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry. The systematic review method (Section 2.3) was selected for this task as it is ideally suited for identifying relevant publications in a fair, repeatable, and thorough manner.

The second study takes into account that there are no experiments where professionals are used investigating the costs and benefits of UML (confirmed in Chapter 4). Using professionals is important as it avoids one of the main criticisms of most controlled experiments in software engineering: results from student experiments may not be representative of developers with industrial skills [44, 45]. In fact, realism is the primary objective of this experiment and therefore drives many of the decisions including the granularity of the UML diagrams and the chosen UML tool. The effects are measured in terms effort, correctness, and code quality. A controlled experiment (Section 2.3) was the natural choice of method for this study, allowing the effect of the manipulation (presence of UML) to be accurately measured – thus increasing confidence in the results. This is confirmed by the obtained statistically and practically significant results.

The two studies are not only highly related, but are also complementary: while the systematic review only aggregates and synthesizes existing knowledge, the controlled experiment contributes to a lacking area exposed by the systematic review.

The topic of the studies is an important one to investigate as it is subject to heated debate. Opinions range from models ultimately succeeding code [4] to models being ineffective [1] (Section 2.2.4). Furthermore, many developers who are positive to UML do not know how to use it most efficiently [46]. Suggestions at effective use of UML do exist, but, they are usually based on advocacy or anecdotal experiences [24]. Answering these questions is therefore important for practitioners and academics alike.

4 A Systematic Review on the Effects of UML during the Maintenance of Object-Oriented Software

4.1 Introduction

This chapter presents a systematic literature review on the effects that the use of standard UML by developers has on the design and maintenance of object-oriented software. A systematic review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest in an unbiased and repeatable manner [47]. As with other disciplines that employ systematic reviews, there are a number of different reasons why one should be undertaken. Some common ones are: to summarize existing evidence concerning a practice or technology; to identify where there are gaps in current research in order to help determine where further investigation might be needed, i.e. to help position new research activities; and to examine how far a given hypothesis is supported or contradicted by the available evidence.

The goal of this review is to, in an unbiased and repeatable manner, aggregate the published empirical knowledge on the topic of the effects of UML on the process and the product during maintenance of object-oriented software. Specifically, we focus on empirical work describing humans working with the standard UML. For example, we are interested in how the presence of UML documentation affects the defect rate (the rate at which a programmer introduces faults into the system), as opposed to the defect rate while not having the UML artifacts. An example of work that would be considered outside the scope is a paper on the efficiency of an algorithm that predicts bottle-necks based on UML models. The results of this review are also then used to identify the holes in the current research so as to steer future research. To accomplish this task, top journals and conferences that publish work on this subject were screened, yielding 23 papers that conformed to the criteria.

The remainder of this chapter is structured as follows: the research method is presented in Section 4.2, the results are presented in Section 4.3, the threats to validity are presented in Section 4.4, and a summary is given in Section 4.5.

4.2 Research Method

This section describes the research questions, the manner in which papers were identified, examined and selected, and the method used to extract and analyze data from papers deemed as relevant.

4.2.1 Research Questions

This review tries to address the general question of what has been empirically learned about the effects that the use of standard UML by developers has on the maintenance of object-oriented software via the six specific research questions presented in Table 4.1. Motivations for each of these research questions are now presented.

Table 4.1: Systematic Review Research Questions

Research Question Code	Research Question
RQ1	What are the costs, risks, and benefits of using UML?
RQ2	What are the most effective ways of using UML?
RQ3	What are the experiences of working with commercial UML tools?
RQ4	Learning curve: how hard is it to learn UML in practice?
RQ5	How is UML being used in the industry?
RQ6	What are the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry?

What are the costs, risks, and benefits of using UML? – The first research question is the most important one in the context of this thesis as Chapter 5 contains a large experiment dedicated to this question. As discussed in Chapter 3, knowing the answers to this question ensures that UML is dealt with in an engineering manner.

What are the most effective ways of using UML? – The second research question stems from the fact that UML is a very large language, much like in programming languages, knowing the constructs alone does not ensure effective use [48].

What are the experiences of working with commercial UML tools? – It is fair to say that UML cannot reach its full potential without adequate tools, and research question three aggregates the reported experience of working with commercial UML tools. Creating high quality models is an investment, and good tools are crucial in order to reduce the costs and maximize the gains of this investment. Further, developers will shy away from poor tools due to the large overhead involved in using them. UML is then only used at the "UML as sketch" level (see Section 2.2.4): quickly and informally, on a simple medium (like paper).

Learning curve: how hard is it to learn UML in practice? – Before a developer becomes productive with a technology she must go through a learning period. During this period it is highly likely that the developer will be less productive, though once the developer learns the technology the hope is that she becomes more productive than prior to adopting the technology [3]. Different technologies have different degrees of difficulty; this chapter investigates the learning curve of UML.

How is UML being used in the industry? – Industrial experience reports and case studies are examined to gauge UML's maturity in industry and the manner in which it is used.

What are the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry? – The purpose of this research question is to aggregate experience on successfully implementing UML in industry.

Furthermore, the insight gained from answering these research questions is used to highlight problems so as to guide future research efforts.

4.2.2 Relevant Article Identification Method

Relevant articles were identified using the approach described in [49]: well-known high quality journals and conferences were selected as well as specialized ones that are likely to contain papers of interest. From these sources relevant papers were then identified using the following strategy:

1. The pool of papers was limited to only those that contained a specific keyword (anywhere in the paper) so that papers with a very highly probability of being irrelevant are excluded.
2. The remaining papers were manually inspected by reading the abstract, and if necessary, scanning the remainder of the paper. A paper was deemed irrelevant if it did not satisfy the specified context.
3. In the case where there was any doubt, at least two researchers reached a consensus.

UML was selected as the specific keyword, a paper not containing this keyword was deemed as highly improbable to being relevant. Next, papers containing the UML keyword were scanned for relevance. The context of interest was specified in Section 4.1 – *What have we empirically learned about the effects that the use of standard UML by developers has on the maintenance of object-oriented software?* For example, grounds for exclusion include:

- The paper did not primarily deal with UML. For example, the paper uses a "UML class diagram" to describe an unrelated concept.
- Not being empirical work. For example, UML-based technique is proposed but is not empirically validated with human subjects.

The focus is on how the *developers' use* of UML affects software development activities and products during software maintenance (the paper must report on results derived from human subjects).

Note that the method of identifying potentially relevant papers via a query-search approach [47] was not used as it was deemed infeasible during a pilot study forcing us to adopt the chosen approach. First, identifying relevant papers using a query proved very difficult as work that is within the scope of our study may be described in many different ways. This, for example, is in contrast to empirical papers with respect to pair-programming, which are easier to identify as the number of total papers on the general topic of pair-programming (identified by the "pair programming" keyword) is manageable. The same cannot be said about UML as it is such a widely-used keyword. Next, to identify relevant papers, we needed to use a large and imprecise query. This resulted in far too many false-positives to filter-out. For example, the *Web of Science* [50] alone returned 4,603 hits (when the search is restricted to publications after 1998). Last, the search engine used by ACM is made to handle only the simplest of queries.

4.2.3 Extent

The following conferences and journals, identified as the most important and likely to contain relevant material, were searched for relevant content:

- The UML/Models Conference (UML/Models)
- International Symposium on Empirical Software Engineering (ISESE)
- International Conference on Software Process and Software Metrics (Metrics)
- International Conference on Software Engineering (ICSE)
- International Requirements Engineering Conference
- IEEE Transactions on Software Engineering (TSE)
- ACM Transactions on Software Engineering and Methodology (TOSEM)
- Empirical Software Engineering: An International Journal (ESE)
- Software and System Modeling (SoSyM)
- Journal of Systems and Software (JSS)

- Information and Software Technology (IST)
- Journal of Software Maintenance and Evolution (SME)
- Journal of Software Practice and Experience (SPE)

The covered timeframe did not have an explicit lower bound since it is implicitly defined by the (first) occurrence of *UML*. The upper bound, on the other hand, was set to the end of 2006.

4.2.4 Data Extraction & Analysis

Once the papers within the scope were identified, the next step was to extract the relevant information (data) from them. This was accomplished by coding the relevant parts of every selected paper with a code corresponding to a research question. This can be considered a top-down approach as we knew what type of data was sought in the papers. Subsequently, a bottom-up content analysis [51] was used to reveal results derived from this data (e.g. papers discussing efficiency of class-diagram metrics when used to assist effort estimation can all be linked). The results of the bottom-up approach are reflected by the subsections of the research-question sections.

4.3 Results

This chapter presents the results for the papers identified as being relevant. First, an overview of the results is given in Section 4.3.1 along with an introduction to every relevant paper. Then, each research question is independently analyzed and discussed in Sections 4.3.2 to 4.3.7.

4.3.1 Overview of Results

Table 4.2 shows the number of relevant papers found in each source, the number of papers that were inspected (these papers contained the UML keyword), and the total number of papers in that source. Out of 6639 papers, 1572 contained the UML keyword and were thus opened and inspected, yielding 23 relevant papers.

Table 4.2: Results by Source

Source	Relevant	Inspected	Total
The UML/Models Conference (UML/Models)	5	369	369
International Symposium on Empirical Software Engineering (ISESE)	4	37	172
International Conference on Software Process and Software Metrics (Metrics)	2	33	294
International Conference on Software Engineering (ICSE)	1	202	1235
International Requirements Engineering Conference	0	114	462
IEEE Transactions on Software Engineering (TSE)	2	146	616
ACM Transactions on Software Engineering and Methodology (TOSEM)	0	37	171
Empirical Software Engineering: An International Journal (ESE)	2	29	214
Software and System Modeling (SoSyM)	0	145	163
Journal of Systems and Software (JSS)	3	143	1211
Information and Software Technology (IST)	3	204	939
Journal of Software Maintenance and Evolution (SME)	1	35	108
Journal of Software Practice and Experience (SPE)	0	78	685
Total	*23*	*1572*	*6639*

Table 4.3 gives an overview of the relevant papers, indicating their source, the research question(s) they contribute to (main contributions to a research question is denoted with a bold *X*), the employed empirical method, the context of the empirical work, the number and type of subjects, the research question(s) addressed in the papers, and the dependent variables. Note that the subjects' background is only specified if it is not software engineering or computer science.

All but one of the papers dealt primarily with either one of the first two research questions (RQ1 or RQ2). The papers that address RQ1/RQ2 are also usually mutually exclusive; this is because these research questions are very orthogonal to each other – loosely speaking, RQ1 looks at comparisons between working with/without UML while RQ2 compares different ways of using UML. Next, RQ5 and RQ6 are addressed by the same papers. This is not surprising as they are both based on industry papers.

Table 4.4 reveals that a large majority of the papers contribute to RQ2 (74%), on second place are RQ1. RQ5 and RQ6 are addressed by a quarter of the papers, last are RQ3 and RQ4, which are addressed by 17% and 13% of the papers, respectively. Table 4.5 gives a breakdown of the papers by experiment design: 75% of the studies employed experiments, followed by case studies (13%), experience reports (9%), and finally one survey (4%).

Table 4.3: Overview of the Relevant Papers

	Source	Research Question 1	2	3	4	5	6	Empirical Method	Context	Num. of Subjects	Research Questions	Dependent Variables
[52]	IST				X	**X**	X	Survey	Industry	131 Respondents	Investigation into the adoption and use of UML in the software development community.	N/A
[53]	ICSE	**X**						2 Experiments	Laboratory	111 Students / 48 Professionals	Are defects in UML models detected by implementers? / How do undetected defects impact the interpretation of the model by different implementers?	Detection Rate, Agreement Measure
[54]	ISESE	**X**	X			X	X	Case Study	Industry	71 Professionals	What are the differences in using UML in legacy vs. new development?	Costs and benefits of using UML in a legacy environment
[55]	JSS	**X**				X	X	Case Study	Industry	9 Professionals	Effectiveness of OO/UML compared to a RAD tool.	Effort, Defect Rate
[56]	JSS	**X**				X	X	Experiment	Industry	12 Professionals	What is more maintainable, OO/UML or procedural, in the context of real-world mission-critical software?	Effort, Change Volume
[25]	UML/Models	X		X		X	X	Experience Report	Industry	N/A	Summarizes Motorola's fifteen years of experience with MDE.	N/A
[57]	TSE	**X**		X	X			Experiment	Laboratory	20 / 78 Students	What is the impact of UML documentation on software maintenance?	Effort, Correctness, Code Quality
[58]	UML/Models		**X**			X	X	Experience Report	Industry	25 Professionals	How to manage use case evolution.	N/A
[59]	ISESE		**X**	X				Experiment	Laboratory	7 Groups of 2-3 Master's Students	What is the most effective way to reverse-engineer code to models?	Questionnaire measuring difficulty, Effort
[60]	UML/Models		**X**	X				Experiment	Laboratory	106 Master's Students	What is the effect of using syntactic modeling conventions on model creation in terms of correctness and effort?	Correctness, Effort
[61]	TSE		**X**		X			2 Experiments	Laboratory	38 / 84 4th year Students	What is the impact of using OCL during object-oriented analysis?	Comprehension, Maintainability, Defect Detection Rate

	Source	Research Question 1	2	3	4	5	6	Empirical Method	Context	Num. of Subjects	Research Questions	Dependent Variables
[62]	ESE		**X**					Experiment	Laboratory	18 last-year students of Informatics	Compares the semantic comprehension of sequence, state, and collaboration diagrams.	Correctness, Effort
[63]	IST		**X**					2 Experiments	Laboratory	31 last-year Students	How UML dynamic diagrams in isolation can support comprehension and which combination of these diagrams improves the understanding of the system most.	Correctness, Effort
[64]	ISESE		**X**					Experiment	Laboratory	40 postgraduate students and members of staff from either the Department of Computer Science or the Department of Psychology	The independent variables investigated in the study were diagram type, user pre-test and post-test preference, individual's cognitive style, text direction, scenario type and question type – with respect to interaction diagrams.	Effort (to arrive to the correct solutions)
[65]	IST		**X**					2 Experiments	Laboratory	76 MIS Students	From the users' perspective – understandability of sequence vs. collaboration diagrams. From analysts' perspective – interested in comparing correctness and completeness of crated sequence and collaboration diagrams.	Correctness, Effort, Perceived Comprehension, Quality, Perceived Ease of Construction
[66]	UML/Models		**X**					2 Experiments	Laboratory	55 / 178 Students	What is the effect of composite states on the understandability of UML statechart diagrams?	Correctness, Effort
[67]	ESE		**X**					Experiment	Laboratory	27 Master's Students	Are systems developed using SORT easier to understand than ad-hoc approaches? / Do documents and techniques derived using SORT permit more efficient and more effective identification of defects than ad-hoc approaches?	Understandability Effort & Correctness; Verifiability Effort, Completeness, Accuracy, Rate
[68]	JSS		**X**					4 Experiments	Laboratory	(39, 15, 9) Students, 8 Professionals	Do stereotypes improve understandability of UML models?	Correctness, Effort, Effort/ Correctness
[69]	METRICS		**X**					1 Experiment, 2 Replications	Laboratory	60 3rd Students, 26 subjects with various backgrounds, and 29 5th year Students	Does a relationship exist between object coupling and understandability / modifiability of OCL expressions?	Understandability, Modifiability

	Source	Research Question 1	2	3	4	5	6	Empirical Method	Context	Num. of Subjects	Research Questions	Dependent Variables
[70]	UML/Models		**X**					3 Case Studies	Industry	6 / 6 / 5 Professionals	The application of effort estimation based on use case points in industry.	Effort
[71]	ISESE		**X**					Experiment & Differentiated Replication	Laboratory	72 / 28 4th year Students	Can early metrics which measure internal attributes of UML class diagrams, such as structural complexity and size, be used as early class diagram maintainability indicators via a prediction model?	Various class diagrams metrics.
[72]	SME		**X**					Experiment	Laboratory	30 final-year Students		
[73]	METRICS		**X**					Experiment & Differentiated Replication	Laboratory	24 / 26 Advanced Students		

Table 4.4: Breakdown by Research Questions

Research Question	Relevant Papers (total 23)
RQ2	74% (17)
RQ1	26% (6)
RQ5	26% (6)
RQ6	26% (6)
RQ3	17% (4)
RQ4	13% (3)

Table 4.5: Breakdown by Empirical Method

Empirical Method	Portion (/23)
Experiment	74% (17)
Case Study	13% (3)
Experience Report	9% (2)
Survey	4% (1)

An Introduction to Each Relevant Paper

An industry survey involving 131 respondents was conducted on the state of UML in industry in [52]. The survey was web-based, contained 32 questions, and the respondents were found via online newsgroups with threads relating to UML. It was mandatory that the respondents be directly involved with UML.

The study in [53] ran two experiments, one involving 111 master's students and one involving 48, professionals investigating the defect detection rate and misinterpretations caused by undetected defects.

The case study presented in [54] looked at the cost and benefits of using UML in a legacy code environment vs. a greenfield environment within the context of a large development project. The project involved approximately 230 people, 100 of whom used UML during development. The developers using UML were in either the group that enhanced existing components or the group that developed software from scratch. The data consisted of interviews that were conducted with 16 members of the project and 55 responses to a questionnaire. The developers were organized in teams, which consisted, on average, of 8 to 10 people.

The case study presented in [55] reports the results of a case study at a company that moved to an OO/UML paradigm from OMNIS 7.3, a Rapid Application Development (RAD) tool. OO/UML was evaluated by (1) reimplementing a part of an existing product and (2) building a new application (with emphasis on code reuse). The case study suffers from a lack of reported detail. First, few concrete measures are given and those that given are very coarse. For example, development productivity is defined as development days per use case with no indications given as to how large a use case is. Next, apart from development effort, no size metrics on of the system are given (expect for the number of use cases). Last, almost no information is given as to how the UML was used or what UML diagrams were used.

The case study presented in [56] examined the issue of OO system maintainability for mission-critical software by conducing a case study where two functionally equivalent systems were developed and maintained: one object oriented (OO/UML) and the other non-object oriented (NOO). This study was conducted in a field setting at a major credit card company in Korea with a credit approval system as the system under study. A

maintenance task was randomly chosen from a pool of 488 change requests. All the developers were software professionals familiar with the system.

Motorola's experience with UML, and MDE in general, is discussed in [25]. Motorola has been using top-down approach to MDE for over 15 years and has shipped many millions of lines of code generated from SDL and UML models.

The study in [57] looks at the costs and benefits of UML via two controlled experiments in two geographical locations (Oslo, Norway and Ottawa, Canada) where senior university students proficient in OO and UML expanded small systems. The objective was to investigate whether the use of UML documentation can make a practically significant difference that would justify the costs. The experiments had one independent variable: the presence (and use) of UML.

The experience report in [58] looks at the question "How do you specify a change to a use case?" The authors tried two solutions: managing use case "deltas" (analogous to code deltas) and the re-specification of the use case with the highlighted change. The authors found that the deltas approach was the most efficient manner of dealing with use case change.

The study in [59] compares two techniques of rebuilding class diagrams from code (existing software). In the first technique, models are created interactively using a model drawing tool (GUI). In the second technique, diagrams are produced by annotating the code with metadata that adds model-relevant information. A tool can then be used to process the data and display the model. To create these models requires human input as source code does not contain all the (semantic) information that a model contains (e.g. is the association an aggregation or a composition?). This study compares these two techniques in order to compare their effectiveness.

In [60], modeling conventions (analogous to coding conventions), focusing on syntactical quality of UML models are investigated. The central idea is that, in the same manner that coding conventions help produce more homogenous and therefore readable code, similar gains may be realized with modeling contentions. The conventions ensure a uniform manner of modeling and to prevent syntactic defects. Examples of conventions are "each message must correspond to a method" and "classes should have low coupling". The experimental task of model creation was carried out in teams of 3 subjects over a six-week period, with a total subject count of 106 master's students. The task of the subjects was to

develop a UML model of the architecture of an information system for an insurance company. The experiment has one factor (the modeling conventions) and three treatments (1) no modeling conventions (the control group); (2) modeling conventions; and (3) tool-supported modeling conventions.

In [61], an experiment and its replication investigate the impact of using OCL on three software engineering activities using UML analysis models: detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes. Essentially, the authors wanted to evaluate whether additional effort and formality associated with OCL brings any tangible benefits in practice. The subjects, fourth year software engineering students, worked on two artificial systems with full UML documentation: UML analysis documents with or without OCL constraints, and with or without seeded defects, questionnaires for the comprehension and maintenance tasks.

In [62], a controlled experiment is performed to evaluate the semantic comprehension of three diagram types, sequence, state, and collaboration, with respect to three (artificial) applications: a simple cellular telephone, a library system and a digital dictaphone (a real-time reactive system). Evaluation is based on answer to multiple choice questions in terms of two metrics: effort and number of correct answers. Subjects consist of 18 last-year students of Informatics.

The study in [63] investigated how individual dynamic diagrams (collaboration, sequence, and state) and paired combinations (collaboration–state, sequence–state, sequence–collaboration) of these diagrams can aid system comprehension via two experiments. The first experiment is a differentiated replication of the experiment reported in [62] and investigates individual dynamic diagrams, the second experiment investigates the paired combinations where the same subjects were used on two new systems. The differentiating factor for the replication is the addition of an independent variable corresponding to the order of presentation of the three application design documents so that any effect due to order can be detected and measured. The dependent variables are effort and questionnaire correctness scores.

In [64], sequence and collaboration diagrams were compared in terms of the effort it takes to arrive at the correct answer. The independent variables investigated in the study were diagram type (sequence or collaboration), user pre-test and post-test preference, individual's cognitive style, text direction, scenario type and question type. The subjects

consisted of 40 postgraduate computer science students and members of staff from either the Department of Computer Science or the Department of Psychology, all of whom had some previous experience with UML diagrams during their studies or work. The subjects had to answer comprehension question based on diagrams for artificial systems. Participants could only continue once they had input a correct answer to a question, an additional measure to try to ensure the information was read carefully.

The article in [65] reports the findings from a controlled experiment where both the comprehensibility and quality of sequence and collaboration diagrams were investigated in two application domains management information systems (MIS) and real-time (RT) reactive systems) from two perspectives: analysts and users. The subjects consisted of 76 MIS students. From the users' perspective, diagram comprehensibility is investigated: "Is there a significant difference between a sequence and a collaboration diagram (in terms of quality as well as comprehensibility) for dynamic modeling of a real-time system?" From the analysts' perspective, diagram quality (i.e. correctness and completeness) is investigated: "Is there a significant difference between a sequence and a collaboration diagram (again, in terms of quality and comprehensibility) for dynamic modeling of a management information system?"

In [66], two controlled experiments investigate whether the use of composite states improves the understandability of UML statechart diagrams. Subjects answered questions testing their understanding of diagrams. The first had 55 Computer Science students from the University of Murcia participated in this experiment. The second experiment, a replication, had 178 Computer Science students from the University of Alicante.

Systematic mapping of UML constructs to code is investigated via an experiment in [67]. Specifically, the experiment evaluates an approach known as Systematic Object-Oriented Refinement and Translation (SORT) with respect to the mapping of object-oriented UML design models to source code, by comparing the effects of different approaches to such mappings (SORT and ad-hoc) on the quality attributes understandability, verifiability, and effort (time). SORT is based on the principle of decoupling refinement from translation by first refining it into a more concrete form at the implementation level of abstraction and second, translating the refined model into a tool comprehensible form (e.g., source code) with the help of patterns (i.e., pre-verified mappings). Using this approach introduces homogeneity in the way that modeling constructs are represented in code – this homogeneity may ease code understanding.

The effectiveness of stereotypes with respect to comprehension is evaluated in [68] via four experiments. Stereotypes are expected to help diagram comprehension as the diagrams are tailed to the problem domain.

In [69], the authors attempt to study if any relationship exists between object coupling (defined through navigations and collection operations), and two maintainability characteristics of OCL expressions: understandability and modifiability.

In [70], the authors conducted three case studies that looked at applying the use case points method for estimating software development effort.

The set of three papers [71-73] is dedicated to investigating how early metrics which measure internal attributes of UML class diagrams, such as structural complexity and size, can be used as early class diagram maintainability indicators via a prediction model. The authors speculate that this may lead to being able to predict the level of correctness and completeness of change tasks the time needed to understand a class diagram before modifying it.

4.3.2 RQ1: What are the costs, risks, and benefits of using UML?

This section discusses papers that contribute to the understanding of the costs and benefits of UML; six such papers were identified [25, 53-57] (introduced in Section 4.3.1). Though all these papers contain information on the question of interest, they vary greatly in nearly every aspect. Also, most papers do not directly address RQ1, and therefore have confounding factors which are discussed in the next paragraph. The papers are then discussed with respect to the following topics: the benefits of UML in terms of communication and documentation, correctness (defects), testing, design quality, and effort. Conversely, the costs of UML are discussed in terms of model construction and maintenance, training, tool support and management support. These topics have been derived using a bottom-up approach, that is, the topics were chosen based on the available information in the papers. Lastly, a discussion on these topics is conducted based on the synthesized results of the five papers.

Confounding factors are present in [55], where three variables are changed simultaneously – adoption of OO technology, adoption of UML, and the adoption of a different testing strategy; [56], where two variables are introduced – OO and UML, thus the results in the paper are mostly reported from that angle; and [25] – the experience report does not exclusively report on UML but rather on model-driven engineering in general.

4.3.2.1 The Benefits

Communication and Documentation Benefits

In [54], both groups were asked whether each diagram type they used had a positive effect on documentation and communication. Both groups felt that these two uses of the diagrams were related in that good documentation was perceived as a prerequisite for successful use of it in communication.

It was found that use case modeling only had a positive effect in greenfield development, developers doing legacy development experienced more problems constructing use cases. In greenfield development, the use cases improved the documentation of the new code by enforcing one structure on all functional descriptions. The authors believe that the legacy group did not see advantages here due to the problems the legacy group experienced during use cases construction: reverse engineering of the legacy code was difficult to fit into the use case framework.

Class and sequence diagrams were both found to have a positive effect on documentation and communication by most respondents. Further, many developers felt that the "class diagrams were the code", meaning that class diagrams facilitated the understanding of the code. Communication within the teams was considered to have improved due to having the UML models as a basis for discussions (e.g. it was easier to come up with suggestions for solutions when the UML diagrams were used in the discussions.) The sequence diagrams were found to be particularly useful for obtaining an overall understanding of the system.

Overall the developers were most satisfied with the effects of class diagrams and least satisfied with the effects of use cases.

In [55] it was reported that communication and documentation were improved, and reliance on individual members of staff members was decreased.

In terms of documentation in [56], the NOO group hardly documented any user requirements and relied on verbal communication. The OO/UML group had more artifacts to maintain than the NOO group, which had only a few mandatory documents.

With respect to communication, during the first three phases (requirements, analysis, and design), the OO/UML group was heavily dependent on UML, whereas the NOO group was forced to utilize direct communication between system analysts and maintainers. Reliance on natural language communication within the NOO group appeared to lead to significant

errors such as ambiguities and missing requirements, which necessitated more frequent communication.

Further, the authors stipulate that “the superiority of OO technology for software maintenance may be due to the usefulness of UML for impact analysis, as employed in the OO group.”

Correctness

In [55] it was reported that quality was increased by lowering the defect rate: “Defect rates appeared substantially lower for the new technology irrespective of reuse levels". A coarse assessment of defect counts suggests that the use of the new technology (OO/UML) improved quality by 73%.

In [56] the authors speculate that “the resulting operation specifications, together with a class diagram that showed a view of the participating classes and interfaces, might have been the reason why the OO group was able to locate the part to be modified more easily, and had fewer costly defects”.

Motorola reports [25] a positive impact from the adoption of MDE where benefits include quality gains. Typical results for quality improvements were a 1.2X–4X overall reduction in defects and a 3X improvement in “phase containment of defects”. The defects are found earlier in the development process where they are less costly to fix. This is largely thanks to model-based code generation and test generation.

In [57] results show that, for complex tasks and past a certain learning curve, the availability of UML documentation may result in significant improvements in the functional correctness of changes as well as the quality of their design.

Testing

In [54], it was ABB’s pursuit of quality assurance that motivated the use of UML-based development. Indeed, it was found that the most positive and noticeable benefits were obtained with respect to using UML diagrams as input to testing. Functional test cases were quicker and easier to define thanks to use cases being used as a basis. The tests were also defined sooner and in a more structured way. Sequence diagrams were particularly useful for ensuring completeness of integration testing while class diagrams were useful for unit testing.

Motorola [25] found that the overall cost of quality decreased due to a decrease in inspection and testing times. Models were used for test generation; this led to the reduction of effort in developing tests, either thanks to the use of abstraction and test generation techniques or the reuse of test models for different test contexts, the improvement of test coverage, and the reduction of defects introduced during the test development process. Further, the use of scenario-based test generation tools yielded an approximately 33% reduction in the effort required to develop test cases.

Design Quality

The project members in [54] were asked whether the UML diagrams had a positive effect on the system design quality. For those developing from scratch, this question was about the extent to which the use of UML had been of help in obtaining a good design. In the case of the legacy group, that started out with code that was not always designed according to object-oriented principles, this question was about whether the different UML diagrams had helped in improving the code structure towards a better and more object-oriented design.

Overall, the developers felt that model-driven development leads to more focus on design. Specifically, the developers felt that use cases had not contributed positively to design; both groups experienced difficulties with deriving classes and methods from use cases in design, though they also found that describing the existing code with use cases led to the identification of some possible improvements in the structure of the legacy code. Nonetheless, the legacy group found that describing the existing code with use cases led to the identification of some possible improvements in the structure of the legacy code. Sequence and class diagrams, however, had positive effects on design.

In [55], the authors note that it was easier to have a larger team working on the same application, reducing the risk of depending on the availability of specific members of staff.

In [57], one of the two experiments also investigated the design of the changes where it was observed that using UML helped to achieve changes with superior design quality, which would then be expected to facilitate future, subsequent changes. Specifically, the authors found that when not using UML documentation, roughly 41% of the solutions contained at least one incorrect change, whereas no incorrect changes were found in any of the solutions that used UML documentation.

Effort

In [55] the results indicated that developing code from scratch using OO/UML was less productive over the development cycle but similar in terms of time to market (compared to the previous used technology): productivity decreased by 142% in the first application and 66% in the second application, in terms of days per use case. Productivity improved by 13.5% when substantial reuse (of 48%) was achieved. Time to market was not significantly affected by the new technology but was greatly improved when substantial reuse was achieved.

In [56] it was found that the OO/UML group required less effort. During the requirements, analysis, and design phases, the OO/UML group used about 70% less time than the NOO group (310 vs. 1020 minutes). During the implementation, test, and deployment phase, the OO/UML group used about 67% less time than the NOO group (215 vs. 660 minutes). The authors note that this was an unexpected result since the OO/UML group needed made more changes to documents than the NOO group. A contributing factor to these results may be the fact that ambiguities within the NOO design documents led to misunderstandings on the part of the maintainers, which, in turn, contributed to some defects.

Overall, Motorola reports [25] a positive impact from the adoption of MDE where benefits include productivity gains. Typical improvements for productivity, measured in terms of equivalent source lines of code, were 2X–8X fold. In terms of effort spent on faults: "…it is not unusual to see a 30X–70X reduction in the time needed to correctly fix a defect detected during system integration testing." They attributed this 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. This was typically done in 24 hours or less, while achieving the same quality with several hundred thousand lines of hand code can easily take one to two months. They note that while the time needed to find the root cause of a defect has been improved in some case and worsened in others depending on the type of the defect: "For example, platform interface issues can be difficult to diagnose since the observed behavior may have no obvious correlation to the model, but subtle logic problems in system behavior are easier to uncover in the model simulation."

Results in [57] indicate that, in the Oslo experiment, the subjects receiving UML documentation spent, on average, used 25 percent less time to solve the tasks than did the subjects without UML documentation. In the Ottawa experiment the no-UML group finished the tasks 2.9% faster. When accounting for model modification, subjects working without UML models finished the tasks faster in both the Oslo and Ottawa experiments (27% and 47.6% respectively).

4.3.2.2 The Costs of UML

In [54], the UML related costs dealt with construction of the diagrams, this was especially true in the case of the legacy group due to having to deal with the legacy systems (reverse engineering was painful). Also, developers had to be trained in UML, the UML-based method, and Rational Rose.

The costs of adopting the new technology in [55] involved buying the tool, training, but, as the authors stress, more importantly, adopters need to invest time and effort into changing their entire development process. This change presents a financial risk to the company.

In [56], the reported costs were additional documentation (UML diagrams) and the fact that a clear separation of roles between analysts and developers of the OO/UML group appeared to add to the dissatisfaction of some who had hoped to learn additional technologies (such as UML) but were restricted to a single task (e.g., programming or testing). Further, while the OO version of the software needed considerably less effort maintain, this came at a cost of more changes to documents than the NOO version. This is not surprising since the OO/UML group had more artifacts to maintain than the NOO group, which had only a few mandatory documents.

In order to realize the gains described in [25], Motorola had to both develop and purchase tools, mange and integrate the various tools, and integrate these into the company via training, adopting the development process, and organizational changes.

In the study reported in [57], it was found that for simpler tasks the time needed to update the UML documentation may be substantial compared with the potential benefits; “Benefits are not likely to be derived if the tasks to be performed lie below a certain level of complexity or if software engineers have not reached a certain level of skill regarding the use of UML models for analyzing the effects of changes, in addition to having received substantial training in UML modeling.”

A study on the effects of syntactic defects in UML models [53] found that most defect types are rarely detected and that some defects types caused a large variation in interpretations. Such defects signal potential misinterpretations due to models, which diminish their overall yield.

4.3.2.3 Discussion

Table 4.6 summarizes the benefits of UML, as reported in the papers (blank cells indicate that the topic was not addressed in the paper). As discussed in the introduction, though the papers are not directly comparable, patterns, as will be discussed, can be observed.

Table 4.6: Benefited from UML?

	Communication and Documentation?	Defects / Correctness?	Testing?	Design?	Effort?
[54]	Yes		Yes	Yes	
[55]	Yes	Yes		Yes	Yes
[56]	Yes	Yes			Yes
[25]		Yes	Yes		Yes
[57]		Yes		Yes	Yes

In terms of communication and documentation benefits, the three papers that discuss the issue all note the positive effects that UML had. This is not surprising as one of the goals of UML was to give developers a common modeling language with which they could communicate more effectively.

Next, four of the five papers discussed the effects of UML on correctness. Two concrete figures are given in terms of effect size that UML has in this area: “improved quality by 73%” and “typical results for quality improvements were a 1.2X–4X overall reduction in defects”. The authors reporting the former figure do not specify why they observed the results, but, two factors are possible: the adoption of a concrete testing strategy and the fact that the developers may have had a clearer understanding of what they were building thanks to the UML documentation. Next, one of the papers observes that the impact analysis was able to be performed much better thanks to the UML documentation. This is understandable as UML diagrams allow system visualization at an abstract level (where the low-level details are omitted).

Testing is discussed in two of the papers discussing UML’s effect on testing. First, in [54] it is interesting to see that the most positive and noticeable benefits were obtained with respect to using UML diagrams as input to testing. In the second paper discussing testing [25], the effects of UML on testing are highly praised. UML’s positive effect on testing can

be attributed to the diagrams being an easier and more structured starting point for test generation. Further, the models allow test generation and ensure completeness.

In terms of UML's effect on design quality, three observations were made on the topic: model-driven development lead to earlier and more focus on design, it was easier to have a larger team working on the same application, and superior design quality was achieved (maybe due to the fact that overly complex design more easily visible on diagrams than in the code).

It seems that UML's effect on effort depends on the context, tools, code generation, and the size of the task. In two of the papers [55, 57] development with UML was less productive, but, in the study that was within an industrial context, it was revealed that thanks to time saved on other activities (error correction), the time to market was the same. It is worth noting that in [57] the tasks were small and the UML overhead was quite large for that system/tasks. In [56] and [25] the productivity gains were quite large: in [56] the UML/OO group used a third of the effort, in [25] the productive, in terms of lines of code, was 2X–8X fold. These interesting results in the former may also be explained also be the simultaneous introduction of OO technology. In the latter, it is probably due to the model-based code generation. Most importantly, it is probable that overall, effort will not be increased by the adoption of UML and as more advanced tools are implemented (e.g. code generation) the gains will increase.

Adoption of UML is not without cost and risks. Costs include training of staff, purchase and integration of tools, and construction of the diagrams. Risk includes misinterpretations of inconsistent and incorrect models, though this can be mitigated with too support and model reviews. As discussed in [54] the construction of the diagrams in legacy systems can be difficult. Further, as discussed in [57], UML can be overkill on simpler tasks. Last, adoption also poses risks for the company from a financial perspective and the fact that the company's development process must be modified.

Considering the five above discussed papers, it seems that the benefits of UML adoption are well worth the costs and the risks. Also, considering the fact that all but one of the papers were not about evaluating the costs and benefits of UML, emphasis is placed on describing the benefits and the costs seem overlooked.

4.3.3 RQ2: What are the most effective ways of using UML?

This section deals with the second research question and looks at empirical work that tried to find effective ways of using UML. Seventeen papers were identified that dealt with the topic. Using a bottom-up approach (see Section 4.2.4), each has been placed into one of the following five categories summarized in Table 4.7.

Table 4.7: Papers with respect to Research Questions 2 Categories

Category	[62]	[64]	[66]	[63]	[65]	[67]	[68]	[60]	[61]	[71]	[69]	[70]	[72]	[73]	[59]	[54]	[58]
Dynamic Modeling	X	X	X	X	X												
Constraints in UML						X	X	X	X								
Measurement and Prediction										X	X	X	X	X			
Reverse-engineering: Code to Diagrams															X		
Use Cases Authoring																X	X

Dynamic Modeling (Section 4.3.3.1) – This section looked at experiments studying and comparing sequence, collaboration, and state diagrams.

Constraints in UML (Section 4.3.3.2) – UML offers various diagram types but provides little guidance as to how these plentiful and powerful language features should be used. Modelers then tend to exploit these degrees of freedom differently. Unfortunately, the lack of uniformity in UML models result in miscommunication between different readers. Industrial case studies [74] and surveys give empirical evidence that individuals use UML in many different ways (even within the same project) and that the number of defects in practice is large. Constraints limit the degrees of freedom provided by the UML; thus, claim to help control its power. This claim is investigated in this section.

Measurement and Prediction (Section 4.3.3.3) – Measurement has a long tradition in natural sciences. Fred S. Roberts, the author of a book on measurement theory [75], points out that that a major difference between a "well developed" science such as physics and some of the less "well-developed" sciences such as psychology or sociology is the degree to which things are measured. Examples of goals of software measurement [76] include the possibility to predict: the error-proneness of a system using software measures from its design phase, the degree of maintainability of a software system, and the amount of effort required to build the software described by that design. Measurement of UML models in particular may allow for assessment of quality at early stages of the software lifecycle.

Reverse-engineering: Code to Diagrams (Section 4.3.3.4) – The topic of reverse engineering code to UML diagrams is important for two reasons: (1) UML will not always be used on a new system and (2) it is a technique that uses UML to help in understanding an existing system.

Use Cases Authoring (Section 4.3.3.5) – This sections looks at how to effectively work with use cases, e.g. during system evolution.

Table 4.8 shows that, with respect to RQ2, the three most investigated areas are Measurement and Prediction (29%), Dynamic Modeling (29%), and Constraints in UML (24%).

Table 4.8: Percentage Breakdown for RQ2

Category	Relevant Papers
Dynamic Modeling	29% (5/17)
Constraints in UML	24% (4/17)
Measurement and Prediction	29% (5/17)
Reverse-engineering: Code to Diagrams	6% (1/17)
Use Cases Authoring	12% (2/17)

4.3.3.1 Dynamic Modeling

Five papers on this topic have been identified: four of the five papers study sequence and collaboration (interaction) diagrams [62-65], two of these also study state diagrams [62, 63], one compares state diagrams to composite-state diagrams [66]. The experiments investigate the advantages and disadvantages of the diagrams, mainly from the point of view of comprehension in terms of effort and correctness.

Table 4.9 summarizes the differences between the experimental material and the mean time spent by the subjects on the experiment for the interaction diagrams. The data was derived from the information in the papers; if no concrete data on this was given in the paper, the sample diagrams were used to derive the numbers. All the experiments were based on artificial systems.

The rest of this section is structured as follows: common results for interaction diagrams are reported in terms of effort and correctness. Next, results unique to each paper are presented. Last, the results are discussed.

Table 4.9: Interaction Diagrams – Experimental Material

	Objects	Messages	Presented in the context of other UML documentation?	Mean Time Spent
[64]	4	26-31	No	205 min
[62]	5-8	4-16	Yes	49 min
[63]	5-8	4-16	Yes	29 min
[65]	7-8	9-11	Yes	30 min

Aggregated Results: Effort

In this section effort regarding sequence and collaboration diagrams is aggregated and the results are summarized in Table 4.10. Considering the effect size, it appears that there is no general trend in the results. Two experiments report sequence diagrams being superior [63, 64], two show that the results are close [62, 65], and one reports the collaboration diagram as superior [65]. Note that [63] is a replication of [62].

A closer look at the experiments reveals that [64] reports effort differently than the remaining papers, i.e. in terms of time required to arrive at the correct solution, and the subjects spent an order of magnitude longer in this experiment than the others. Further, only this result is statistically significant.

Table 4.10: Interaction Diagrams: Effort

	Sequence (min)	Collaboration (min)	Effect Size	Significant?
[64]	194.01	216.77	11.7%	Yes
[62]	50.02	48.28	-3.5%	No
[63]	27	30.47	11.4%	No
[65] - MIS	28.450	26.000	-9.4%	No
[65] - RT	22.473	23.052	2.5%	No

Aggregated Results: Correctness

In this section correctness regarding sequence and collaboration diagrams is aggregated. The results are summarized in Table 4.11 in terms of the percentage correctly-answered questions for each diagram type, the effect size, and statistical significance. Overall, there is no general trend between the diagram types in terms of correctness. Two of the results show that both diagrams generate the same number of correct answers [62, 65]. One of the results indicates that sequence diagrams are more effective [63] and one of the results indicates that sequence diagrams are less effective [65]. Note that [63] is a replication of [62].

Table 4.11: Interaction Diagrams: Correctness

	Sequence	Collaboration	Effect Size	Significant?
[62]	72.2%	71.1%	1.5%	No
[63]	71.3%	66%	8.1%	No
[65] - MIS	85%	84.4%	0.7%	No
[65] - RT	46.3%	66.3%	-30.2%	Yes

Other Observations

This section discusses results unique to the papers, i.e. those that cannot be aggregated across the papers. A summary of the topics and the finding is presented in Table 4.12.

Table 4.12: Summary of Other Dynamic Modeling Observations

	Topic	Finding
[64]	Diagram type preference, text orientation, and scenario familiarity.	Diagram preference positively affects performance. Text orientation does not matter. Scenario familiarity positively affects performance.
[62]/[63]	State Diagrams – Effort/Correctness	Are more time consuming, but yield more correct answers.
[63]	Compared paired combinations of diagrams – Effort/Correctness.	Sequence–State pair gave the best results in terms of correctness, no significant difference in terms of effort.
[65]	Perceived comprehensibility.	MIS – sequence diagram were 5.4% less comprehensible. RT – sequence diagrams were 8% more comprehensible.
[65]	Perceived ease of construction.	MIS – sequence diagrams were 6% more difficult RT – sequence diagrams were 14% less difficult
[65]	Quality of constructed diagrams.	MIS – collaboration diagrams of better quality than sequence diagrams (9.4% effect size) RT – no significant difference in quality of diagrams created in RT systems (the sequence diagrams were 5.3% better)
[66]	Comparing state diagrams to composite-state diagrams.	Conflicting results.

Results particular to [64] show significant differences for several variables, including diagram type preference, text orientation, and scenario type. In the case of diagram preference, subjects performed better using the diagram type they preferred. No significant effect on user performance could be attributed to text direction for either sequence or collaboration diagrams – thought it is suspicious that this should make a significant difference in the first place. In terms of the effect that scenarios play on user performance, scenario familiarity and complexity influence how well diagrams are understood.

The study in [62] (and replicated in [63]) also compared state diagrams to the interaction diagrams in terms of effort and correctness. Such a comparison is not balanced as state diagrams are complementary to interaction diagrams; they do not show the same information. Nonetheless, in terms of effort, the state diagrams showed to be more time consuming than the sequence and collaboration diagrams in both [62] and [63],

respectively, 1.7% and 15.9% more time was spent on than sequence diagrams. In terms of correctness state diagrams were found to provide the highest number of total correct answers in both [62] and [63], respectively, 4.6% and 0.9%.

The second experiment in [63] compared paired combinations of diagrams (sequence, collaboration, and state). The authors found that the Sequence–State pair gave the best results in terms of comprehension – about a 5% increase over the Collaboration–State pair score. There were no statistically significant differences in terms of effort. The Sequence–Collaboration pair gave the worst results.

The results specific to [65] are the perceived comprehensibility (post-test questionnaire), perceived ease of construction (post-test questionnaire), and the quality of constructed diagrams. In the case of the MIS system the perceived comprehensibility showed the sequence diagrams to be 5.4% less comprehensible than collaboration diagrams (statistically insignificant). In the case of the RT system, the insignificant results showed that sequence diagrams were 8% more comprehensible.

The perceived easiness measuring difficult to build the diagram showed that for the MIS system the sequence diagrams were 6% more difficult (statistically insignificant). In the case of the RT system, the insignificant results showed the sequence diagrams being 14% less difficult.

With respect to diagram quality, where the 'quality task' was to construct a diagram, in the case of MIS, analysts create collaboration diagrams of better quality than sequence diagrams (9.4% effect size), but there is no significant difference in quality of diagrams created in RT systems (the sequence diagrams were 5.3% better). Irrespective of the diagram type, more correct diagrams are created in MIS applications than in RT applications. The authors do not provide an explanation for this nor do they clearly specify how they obtained these figures apart form "quality was measured by the correctness of the created diagrams".

Last, in [66], the "understandability efficiency", defined as correct answers given by the subjects divided by the time spent on answering the questions related to an UML statechart diagram, the difference is statistically significant (58% higher for composite states). The authors argue that this indicates that the diagrams with composite states are superior. In the replication, the results were 17% lower for composite states. Thus, using the "understandability efficiency" metric, the results in the replication showed contrary results.

Here the authors discredit the results by blaming "the lack of experience of the subjects working with this kind of UML diagram was a key factor in obtaining these results."

Discussion

The five discussed papers [62-66] empirically investigated the use of UML diagrams for dynamic modeling: sequence, collaboration, and state. Four of these held experiments comparing interaction diagrams, the results in terms of effort and correctness are summarized in Table 4.10 and Table 4.11. No overall trends in terms of effort or correctness were observed.

Only one of the effort results was statistically significant [64], this results is also of practical significance, but, the diagrams were informal and not in the context of other documentation (e.g. use cases and class diagrams). The subjects involved people with a psychology background, which may not have had the adequate background knowledge to participate in the experiment.

The experiment in [62] was replicated in [63] (with 50% more subjects), and gave inconsistent results for both effort and correctness, giving further grounds to believe that the lack of statistical significance is in fact due to the results being due to chance (and not a Type II error). Further, the application types are not rigorously defined or distinguished.

The main weakness with the study in [65] are the subjects. Unfortunately MIS students are not familiar with RT systems, thus, this fact must be taken into account when looking at the results related to the RT system. The fact is pointed out by the authors: "our participants lacked adequate skills and training for comprehending real-time applications". One may question whether the results also on the MIS system would be the same if the subjects had proper software engineering training.

The lack of an emergence of clear trends in either the effort or the correctness may be due to the fact that the experiments are not asking the right questions. The decision as to which diagram should be used, sequence or collaboration, depends on the properties that the diagram should highlight. This is an engineering decision where tradeoffs must be made and where professional experience comes into play. For example, sequence diagrams focus on time-oriented visualization that is particularly helpful in real-time systems. In fact, sequence diagrams have their roots in message sequence charts [77] where a primary goal was to support real-time system visualization. Collaboration diagrams, on the other hand, focus on object structure and, during analysis and design, help validate and discover

associations between classes. With proper tool support, the fact that the diagrams are isomorphic is helpful as the developer can switch views depending on the activity they wish to perform.

Thus, instead of asking "What is the best sequence, collaboration, or state?" it would be better to acknowledge the fact that these three diagrams should be used in different situations. This can also be seen in [64] where the results show that scenario familiarity plays a significant effect on user performance. Given this, future experiments should ensure that they use subjects with proper training and a proper educational background, and focus on the following questions instead:

- What level of details should sequence/collaboration diagrams contain? Too much detail and the developers are overwhelmed and may find it easier to work with code (diagrams get abandoned). Further, diagram detail has direct implications on diagram-code synchronization.
- Anecdotal evidence suggests that developers find little use in diagrams that are "too small" or "too large". This should be investigated to see with what kind of diagrams, with respect to size, most developers can work most effectively with.
- What is the most effective way of combining and using the different UML diagrams with the interaction diagrams? This question should be asked in terms of both, the presence of other diagrams and the user interface. For example, how much more useful are interaction diagrams when class diagrams are also present? And, how should developers interact with the diagrams (UML tools) to maximize efficiency?

In terms interesting results reported in *Other Observations*, the experiment in [62] and replication in [63] also compared state diagrams to the interaction diagrams. It is not clear why the authors treated the state diagram as being equivalent to the sequence/collaboration diagrams, as they are a different view on the system, focused on conveying different information. This is repeated in the second experiment in [63] where the interaction/state diagram pairs were also compared to interaction/interaction diagram pair (sequence-collaboration).

In [66] state diagrams with and without composite states are compared in an experiment and a replication. It is difficult to interpret the results from the paper as the main metric is measured by the authors using their "understandability efficiency" metric defined as

correct answers given by the subjects divided by the time spent on answering the questions related to an UML statechart diagram. Correctness and effort data is not provided. The authors seem biased: “Based on previous experiments … and on our intuition and experience working with UML statechart diagrams, we think that the answer to this question should be a ‘yes’, especially when the person that is trying to understand the UML statechart diagram is used to working with this modeling language and this kind of diagram.” Further, diagram sizes are not reported (though the authors claim the models were “simple”). The diagrams are not in the context of other UML artifacts.

4.3.3.2 Constraints in UML

Four papers have been identified as dealing with a form of constraint; in [68] stereotypes are evaluated. Stereotypes are used to create a new model element by introducing new semantics to an existing model element. The new element, targeted for a particular problem domain, has the same structure as an existing element, but with additional constraints, a different interpretation and icon, and different treatment by code generators and other back-end tools. In [60], modeling conventions (analogous to coding conventions), focusing on syntactical quality of UML models are investigated. OCL, a language specifically designed to add formality, is investigated in [61] in terms of detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes. Last, effects of systematic mapping of UML constructs to code with respect to understandability, verifiability, and effort is investigated [67]. The papers are summarized in Table 4.13.

Table 4.13: Summary of Constraint-based Experiments

	Constraint Type	Target of Measurement	Type of Measure	Results
[68]	Stereotypes	Understanding of stereotyped / non-stereotyped diagrams.	Questionnaire	Stereotypes do seem to help make comprehension diagram in a practically relevant manner.
[60]	Modeling Conventions	Quality of created diagrams with and without modeling conventions.	Defect Rate	Decreased defect density is attainable at the cost of increased effort.
[61]	OCL	UML analysis documents with OCL/informal constraints.	Comprehension, Maintainability, Defect Detection Rate	Modest benefits at a cost of a significant learning curve.
[67]	UML to Code Mappings	UML docs and code built with SORT and ad-hoc	Understanding Time/Correctness, Verification Time/ Completeness/ Accuracy/Rate	Significantly better understanding of the SORT system in terms of correctness and completeness. Verification tasks on the SORT system were performed significantly better in terms of verification rate, correctness and completeness.

First, the effectiveness of stereotypes with respect to comprehension is evaluated in [68] via four experiments. Stereotypes are expected to help diagram comprehension as the diagrams are tailored to the problem domain. Each experiment was a paired comparison design where both groups had to provide answers to a questionnaire with 12 questions. Dependent variables consist of the correctness score, the effort, and a ratio of Effort/Correctness. The treatments were model types, with two possible values—stereotyped model and non-stereotyped model. The study reported here was carried out using software engineering (and related) students and software engineering industry professionals. Experiments #2 (modified order of object presentation) and #3 (stereotypes were not represented by graphical icons) are differentiated replications, experiment #4 is a replication. The stereotypes are targeted to the telecom domain. The designs contain 30 objects in collaboration diagrams each and 11 (models with stereotypes) or 14 (models without stereo types) classes in class diagrams.

Table 4.14 summarizes the experiments along with the results in terms of effect size (being reported with respect to the non-stereotyped results). Stereotypes were found to increase correctness in all four experiments, with half of the differences being statistically significant. Stereotypes also decreased the necessary effort in 3 of the 4 experiments, though those differences were not statistically significant. Using statistical significance testing for three experiments and a qualitative analysis for the industrial experiment, the authors also found that stereotypes with their graphical representation as icons improve the

understanding of UML models more than stereotypes represented as textual adornments of model elements. Thus, the results show that stereotypes do seem to help make comprehension diagram practically relevant manner.

Table 4.14: Summary of the 4 experiment results in [68]

Experiment	Subjects	# of Subjects	Correctness Difference (Significant?)	Effort Difference (Significant?)
1	Students	39	52% (Yes)	-25% (No)
2	Students	15	69% (Yes)	0% (No)
3	Students	9	45% (No)	-20% (No)
4	Professionals	8	131% (No)	-24% (No)

The paper does leave a few unanswered questions. First, what is the cost associated with using these stereotypes? Next, the metric ratio of Effort/Correctness offers questionable value; we thus chose not to include it in the results discussion. Last, the correctness scores seem low; the mean ranges from 3.25 to 7.50 (out of 12).

Next, the idea of using syntactical modeling conventions is investigated in [60] with respect to defect density and modeling effort. The results show that defect density is reduced by 47.6% in (2) and 54.2% in (3), but the results are not statistically significant. The difference in effort is statistically significant and 12.7% greater in (2) and 64.3% greater in (3). Thus, decreased defect density is attainable at the cost of increased effort when using modeling conventions.

It is worth noting that the effort data is probably not accurate as these experiments were executed with no tool support or very crude tool support.

In [61], an experiment and its replication investigate the impact of using OCL on three software engineering activities using UML analysis models: detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes. The results show that, in terms of defect detection, comprehension, and maintenance, the gains are modest (a 5 to 7 percent increase) at a cost of a significant learning curve. The results are statically significant for comprehension in the first experiment, and for all three dependent variables in the replication.

Unfortunately, an aspect that is not studied is benefit of having all these constraints formally defined during system maintenance: the possibility to dynamically check the constraints. Using automatic dynamic constraint enforcement, developers are instantly

notified that they did something that invalidates the integrity of the system. Thus, the benefits of using OCL shown here may only be partial.

Systematic mapping of UML constructs to code is investigated via an experiment in [67]. The experiment looks at two questions: (1) Are software systems developed with SORT easier to understand than software systems developed using ad-hoc approaches? (2) Is the identification of defects more efficient and effective in documents and techniques derived using SORT than ad-hoc approaches?

In order to evaluate (1) the following are measured: Understanding Time, the time needed to understand the system in order to complete a questionnaire; and Understanding Correctness, the number of correctly answered questions. In order to evaluate (2) the following are measured: Verification Time, the time needed for identifying and documenting defects; Verification Completeness, the completeness or effectiveness of the verification task, captured by the number of defects to be found; Verification Accuracy, the accuracy of the verification task captured by the number of places identified correctly; and Verification Rate, the verification rate or efficiency.

Table 4.15: Summary of the results in [67]

	Effect Size	**Significant**
Understanding Time	-5.2%	No
Understanding Correctness	66.7%	Yes
Verification Time	-1.4%	No
Verification Completeness	55.6%	Yes
Verification Accuracy	89.1%	Yes
Verification Rate	62.5%	Yes

The results are summarized in Table 4.15, with the effect size reported with respect to the ad-hoc results. The results of the empirical evaluation show that applying the SORT technology in system development has a significant impact on understandability and verifiability. Specifically, the subjects had a significantly better understanding of the SORT system in terms of correctness and completeness. Further, subjects performing verification tasks on the SORT system performed significantly better in terms of verification rate, correctness and completeness.

In summary, four studies looked at different ways of adding constraints model-based constraints. Three of the four investigated methods suggest practical gains. First, domain-targeted stereotypes were found to significant and practical improve correctness. Next, although the results in the syntactic modeling conventions experiment were not statistically

significant, they, not surprisingly, suggest a large improvement in the (syntactic) model defect rate. Thus, they should be integrated into a tool enabling an automated and user-friendly manner to minimize the effort required for their use. Subsequently, employment of OCL does not seem to justify its cost, at least without tool-support for dynamic constraint enforcement: natural language may be sufficient. Last, SORT, the systematic UML to code mapping method significantly and practically helped in understanding and verification activities with no impact on effort. Thus, based on these four studies, modelers can expect tangible benefits by taking advantage of this space.

4.3.3.3 Measurement and Prediction

This section examines empirical work in terms of UML model measurement and prediction on the following topics: class diagrams, OCL, and use cases. Last, conclusions are drawn on the topic.

Class Diagrams

A set of three papers [71-73] is dedicated to the investigation of how early metrics which measure internal attributes of UML class diagrams, such as structural complexity and size, can be used as early class diagram maintainability indicators via a prediction model. The authors speculate that this may lead to being able to predict the level of correctness and completeness of change tasks and the time needed to understand a class diagram before modifying it.

All three papers looked at the same metrics, summarized in Table 4.16. The dependent variables measured understandability and modifiability (via correctness and completeness). The understandability tasks required subjects to answer a questionnaire while the modifiability tasks required the subjects to modify the class diagrams according to requirements. Experimental materials consisted of class diagrams from different application domains.

Table 4.16: Class Diagram Metrics

Metric	Description
Number of Classes (NC)	The total number of classes.
Number of Attributes (NA)	The total number of attributes.
Number of Methods (NM)	The total number of methods.
Number of Associations (NAssoc)	The total number of associations.
Number of Aggregations (NAgg)	The total number of aggregation relationships (each "whole-part" pair in an aggregation relationship).
Number of Dependencies (NDep)	The total number of dependency relationships.
Number of Generalizations (NGen)	The total number of generalization relationships (each "parent-child" pair in a generalization relationship).
Number of Generalization hierarchies (NGenH)	The total number of generalization hierarchies.
Number on Generalization hierarchies (NAggH)	The total number of aggregation hierarchies (whole-part structures) .
Maximum DIT (MaxDIT)	It is the maximum DIT value obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy.
Maximum HAgg (MaxHAgg)	It is the maximum HAgg value obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the longest path from the class to the leaves.

The results from the five experiments, summarized in Table 4.17, obtained using multivariate analysis, fail to show an overall trend. The class diagram characteristics that seem to have the largest impact on diagram understanding and maintainability, i.e. they were confirmed by at least two studies, seem to be Number of Attributes and Number of Methods.

Table 4.17: Summary of the Results (X = Significant Result)

	NC	NA	NM	NAssoc	NAgg	NDep	NGen	NGenH	NAggH	MaxDIT	MaxAgg
[73]		X						X	X		
[71]		X	X			X	X			X	X
[72]			X	X							

There are a number of issues with these papers. First, it is important to note that all three papers are linked by at least one author. Second, the experiments were not held in the context of other UML documents. Third, some metrics, like Number of Generalizations and Number of Generalization hierarchies, are perhaps too closely related: they measure very similar things.

OCL

In [69], the authors attempt to study if any relationship exists between object coupling (defined through navigations and collection operations), and two maintainability characteristics of OCL expressions: understandability and modifiability. The authors

performed an experiment along with two replications involving the following subject groups: 60 third year CS students, 26 subjects with various backgrounds, and 29 fifth year software engineering students. The considered metrics, along with the summarized results are presented in Table 4.18, and show the number of experiments where a statistically significant trend was visible. These preliminary results show some significant results (shown in bold in the table) and should be investigated further.

Table 4.18: Summary of the Results for [69]

Metric	Understanding	Modification
Number of Navigated Relationships (NNR)	0/3	**1/3**
Number of Attributes referred through Navigations (NAN)	0/3	**1/3**
Number of Navigated Classes (NNC)	**1/3**	0/3
Weighted Number of Collection Operations (WNCO)	**1/3**	0/3
Depth of Navigations (DN)	0/3	**1/3**
Weighted Number of Navigations (WNN)	0/3	0/3
Number of Explicit Iterator variables (NEI)	0/3	0/3
Number of Explicit Self (NES)	0/3	**1/3**
Number of Comparison Operators (NCO)	0/3	0/3

Use Cases

In [70], the authors conducted three case studies that looked at applying a method for estimating software development effort based on use cases: the use case points method. The three industrial projects in finance, CRM, and banking consumed 3000 to 4000 person hours and involved 5–6 developers. The paper provides a detailed description of the method used and experiences from applying it.

The results, summarized in Table 4.19, show that estimates derived using the use case points method were close to the expert estimates made by senior members of the development projects. The authors found that while the use case points method appears to reduce the need for expert knowledge in the estimation process, applying the method in practice is not straightforward. For example, the choice of structure for the use case model has an impact on the estimates. Further, an important prerequisite for the method is that the use cases of the system under construction have been identified at a suitable level of detail.

Table 4.19: Estimated vs. Actual Results in [70]

Project	Expert Estimate	Use Case Estimate	Actual Effort
A	2730	2550	3670
B	2340	3320 / 2730	2860
C	2100	2080	2740

Discussion

This section looked at UML model metrics used for measurement and prediction. In terms of class diagram metrics, the characteristics that seem to have the largest impact on diagram understanding and maintainability, i.e. they were confirmed by at least two studies, seem to be Number of Attributes and Number of Methods. Only one study exists on the topic of using OCL metrics for understanding and maintainability predictions, making it too early to draw conclusions. A series of case studies on the topic of using the use case points method for effort estimation reveals that the method does reduce the need for expert knowledge in the estimation process.

While these results are interesting, more work is needed in this area by different groups, on different projects and project types to gain greater confidence in the metrics and methods.

4.3.3.4 Reverse-engineering: Code to Diagrams

The study in [59] compares two techniques of rebuilding class diagrams from code (existing software). Specifically, the experiment looked at three questions (related to dependent variables in Table 4.20): (1) Is it easier to annotate the code or to edit the diagrams? (2) How do the recovered diagrams differ in their quality? (3) In which case do the documentation guidelines provide more help?

The experiment design was: two groups, one factor and two treatments. The subjects performed the same re-documentation task with both techniques in two different orders. The subjects were split in 7 groups of 2 or 3 students in their final year of a Computer Engineering master degree. The dependent variables were derived from answers (expressed on a 1 to 5 ordinal scale) to questions summarized in Table 4.20.

Table 4.20: Results for [59]

	Dependent Variable	**Annotation**	**Drawing**	**p-value**
(1)	Difficulty in splitting the system into multiple views	**4**	**2**	**0.0028**
	Difficulty in attribute selection for display in the diagram	**3**	**1**	**0.0152**
	Difficulty in specifying the relationships	**4**	**2**	**0.0040**
	Effort in generating the diagrams	10	5	0.5653
(2)	Level of satisfaction with the default diagrams	**2**	**4**	**0.0476**
	Satisfaction with final result	3	4	0.2013
(3)	Training usefulness	4	4	0.5229
	Process conformance	3	4	0.0967

The results reveal that the drawing editor approach resulted to be the most preferred and usable one, with no penalty to the quality of the resulting diagrams. This was mainly thanks to the GUI; it supported the specification of multiple system views in an intuitive way. The drawing approach made it easier to specify which class attributes (fields and methods) to display, as well as to the inter-class relationships. Also, the default diagrams produced by the drawing tool were judged to be substantially better than those obtained by the annotation approach.

4.3.3.5 Use Cases Authoring

Challenges and solution to use case authoring challenges during maintenance are discussed in two papers: [58] discusses way of managing change during use case evolution and [54] looks at the problem of introducing use cases in a legacy system.

The context in [58] was an enterprise software system with a team of 25 people worked on a system for 18 months. The application was developed using J2EE and the Oracle Suite. The requirements were managed in a word processor, and the system's scope was managed with a spreadsheet. The complete application had a total of 35 use cases and 15 iterations. Iterations were on average 5 weeks long. Each iteration included, either fully or partially, around 10 new or modified use cases.

Two types of the "highlighting change" approaches were tried by the team with no success: using the "Track Changes" feature of the word processor and manually highlighting changes in the document. Both were found to be too inefficient due to the amount of information produced being too great.

The authors solved the problem with *use case deltas*, defined as any change in a use case description that results from the addition, change or deletion of functionality described in a use case. A delta includes the critical information that needs to be specified. A delta specifies three pieces of information: the use case it relates to, a description of the change to the use case, and the source that triggered the change.

The authors then go on to describe their system of using the deltas: a Delta Threshold was set where, if changes to the use case fell below the threshold, it was to be added a delta. If the threshold was exceeded, the use case was to be rewritten. Two types of use cases were defined to deal with the proliferation of deltas: project and iteration. Project use cases are persistent. Iteration use cases, on the other hand, are temporary. An early merging mechanism was used to update the project use cases. The authors found that merging at the

end of every iteration seemed to be the reasonable approach, and a good mitigation for the proliferation of deltas. Last, to help with delta management, a mapping tool was used that represented a description of the delta tree paths.

Next, the case study in [54], first introduced in Section 4.3.2, deals with the topic of applying UML in legacy development. The results of the case study are relative to developers that worked on a greenfield project. In the study, the developers were asked whether the UML diagrams had a positive effect on the various activities of system development. Generally, the authors found that extracting use cases from a legacy system was very difficult. Specifically, identifying and documenting actors was considered a rather easy activity on the new project, but was more difficult in legacy development – the different legacy subsystems often lacked appropriate descriptions of their interfaces at the outset of the project. Next, identifying and documenting use cases was much more difficult in legacy development. Further, the lack of documentation and traceability in the legacy code, in the cases where the existing code was unfamiliar, made it difficult to relate existing functionality, described as use cases, to code.

The developers were also asked their opinion of how the use case diagrams impacted various activities; this is summarized in Table 4.21. In terms of design, use cases modeling had not contributed positively to design for either of the two groups. The authors felt that this is because both groups experienced difficulties with deriving classes and methods from use cases in design. Though, in the case of the legacy development subjects, the interviews revealed that describing the existing code with use cases led to the identification of some possible improvements to the legacy code. With respect to testing, the effects of applying UML-based development were the most noticeable. The developers felt that use case diagrams contributed very positively to the testing (see Section 4.3.2.1 – Testing). Last, in terms of the diagrams effect on documentation and communication as seen by the developers, use case modeling had a positive effect for those developing from scratch. Most of the legacy developers had, however, not experienced positive effects. The authors speculated that this was due to more problems being experienced in constructing the use cases and the use cases made by legacy developers being often too detailed to give an overall understanding of the functionality. It is worth noting that in that case study, overall, the respondents were least satisfied with the effects of use cases (compared to sequence and class diagrams).

Table 4.21: Developers' Opinion on Usefulness of Use Cases for Various Activities

	Design	Testing	Documentation and Communication
Legacy	No	Yes	Yes
Greenfield	No	Yes	No

4.3.3.6 Summary

This section reports on papers that contributed to RQ2: What are the most effective ways of using UML? Seventeen papers were examined in terms of five categories. First, dynamic modeling was examined in Section 4.3.3.1, with five papers contributing to the topic. Most of the papers compared sequence diagrams to collaboration diagrams in terms of effort and correctness for diagrams of various complexities and in different problem domains. The aggregated results failed to show general trends, this is probably due to the wrong kind of question being asked. Given the outcome, the section concludes with suggestions for future experiments, e.g. evaluating the optimal amount of detail that should appear on a sequence diagram.

Next, the possibility of increasing UML's effectiveness via different types of modeling constraints was investigated in Section 4.3.3.2. Three types of modeling constraints were investigated in four papers: stereotypes, modeling conventions, OCL, and UML to code mappings. All but one of these (OCL) offered practical gains, indicating that this is an area of which modelers should be taking advantage of.

Attempts have also been made at exploiting UML models for predictive purposes; this is investigated in Section 4.3.3.3. Predictive models based on metrics for three UML constructs are investigated: class diagrams, OCL, and use cases. From a practical standpoint, only the use case method shows enough maturity to be used in practice, whereas the other results show potential but more research is needed as the method is still immature.

In industry most projects are not greenfield projects, meaning that developers have to deal with existing (legacy) code. This means that often UML will have to be incorporated in a project where it was not used from the start. Given this, it is important to find efficient method of deriving UML diagrams from existing code. This is investigated in terms of reverse engineering class diagrams in Section 4.3.3.4. The authors compare two methods: a code-centric method and a GUI-based method. They find that the GUI-based method is superior. The practical significance of this is that tool developers should incorporate this functionality into their UML environments.

Last, issues when dealing with use cases are looked at in Section 4.3.3.5. Two challenges are discussed: reverse engineering of use cases from legacy systems and dealing with use case deltas.

4.3.4 RQ3: What are the experiences of working with commercial UML tools?

In the set of 23 papers identified by the systematic review, four papers discussed experiences with tools, one has done so extensively.

Motorola's extensive experience with tools is documented in [25]. Note that since the advantages of using the tools have already been discussed in Section 4.3.2, this section focuses on the problems encountered. First, code generated by Motorola's in-house tool (Mousetrap) had fewer defects than vendor tools (though information on which tools and vendors in particular is omitted). Mousetrap detected model problems that most vendor tools could not catch and it generated code that was more complete and of higher performance. They attributed this to optimization techniques and "the luxury of knowing the target platform and being able to customize the code generation for it".

Next, the lack of common tools is outlined as another problem, citing "the inability to completely transfer models between tools, use of vendor-specific extensions, lack of complete UML support, and code generation support for different subsets of UML". This is a serious problem as no single tool supports all necessary features. The authors give an example: "during the transformation of models to code, how is traceability handled if a separate traceability tool is used?"

Last, Motorola found that the MDE tools often had scalability issues: problems with the abilities to load, save, compare, and generate code from large models. Further, performance issues with the generated code were difficult to address due to limited ability to customize the code generation constructs.

The SDMetrics tool [78] was used in [60] to ensure modeling convention adherence. The authors found that the difficulty of performing the task with tool-supported modeling conventions was about 10% higher than for the group that did not use the tool. The authors hypothesized that that integrating adherence checks into UML development tools would decrease this extra effort and result in higher adherence levels, thanks to the shorter feedback loop.

The UML student-experiments in [57] used two modeling tools: TAU [79] and Visio [80]. The authors explain that despite the students having received extensive training in TAU, the level of subjects' dissatisfaction with the tool led to the use of a simpler tool, Visio, in the second experiment. The identified areas of improvement were better support for making changes to models, model-consistency checking, and functionality for keeping models and code synchronized.

The Omondo UML tool [81] was used in [59] for reverse engineering class diagrams from code. The authors found that the tool is not currently able to support this activity in a satisfactory way. Specifically, two points are raised with respect to structural views (views that address a specific subset of the system): the tool does not distinguish between the code editing and the view editing activities, and, view editing should not result in code modifications, such as the insertion of the tool's special-purpose comments.

Given the fact that the systematic review consisted of 23 papers, it is surprising to see that only four of these discussed experiences with UML tools. This signals that more emphasis should be placed on this area so that tool vendors see where they should focus their efforts. Next, basic functionality like model-consistency checking (reported by [60] and [57]), model/code synchronization, and good model-editing support has not been reached (reported by [57] and [59]). This is disappointing as without this (minimum) functionality the overhead of using UML tools for developers increases. Further, it makes it difficult to reach higher goals like code and test generation. Last is the issue of tool interoperability, since developers require different features from different tools, vendors must focus on tool interoperability and IDE integration. Such interoperability creates an ecosystem where the sum (of the individual tools) is greater than its parts.

4.3.5 RQ4: Learning curve: how hard is it to learn UML in practice?

In the two experiments on the impact of UML documentation on software maintenance [57] (introduced in Section 4.3.2), the authors argue that the benefits of using UML only became visible after the developers went through a learning curve. This is based on the observation that the UML group outperformed the no-UML group in terms of the functional correctness of changes as well as the quality of their design on the last task of each experiment. Specifically, the gain in correctness in the first experiment was in 43% on the final task (versus -2% – 10% on the previous tasks), in the second experiment correctness increased by 42% on the last task (versus 0% in each of the previous tasks).

Design quality was only measured in the second experiment and it was found that on the last task there were 50% more correct changes made by the UML group (with no differences on previous 3 tasks). The authors hypothesize that the subjects were only effective after gaining experience using UML on a few tasks first.

OCL's learning curve was seen in [61] (introduced in Section 4.3.3.2) where OCL's effect on the detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes was investigated via two experiments. The results showed that modest gains (5–7 percent) were only possible after "substantial, thorough training to the experiment participants".

The authors came to this conclusion after observing that in one of the experiments there were large performance improvements on the subjects' second attempt of answering questions in the presence OCL (over the first attempt). The results of the first experiment motivated the authors to administer more training prior to the second experiment. The second experiment showed that using OCL had significant positive effect (for all but the first task), whether taking all observations into account, or considering data from each attempt.

Unfortunately the authors do not give an idea as to how much training is necessary before become competent at using OCL.

Developers' feelings on the UML training they received is gauged in [52], an industry survey involving 131 respondents. The results showed that a third of the respondents felt that they have not received adequate training, a fifth did not have an opinion, and almost half felt that the training was adequate. Of the respondents, 41% received some type of formal training. Seeing that only less than half of the respondents found the training was adequate signals a problem with the training-infrastructure.

Even though only three papers discussed UML's learning curve, they all consistently showed that UML is not trivial to learn. This is an important fact to acknowledge so that UML is not thrown at developers who have not been adequately prepared first via serious training. This revelation makes sense; first UML's specification is not small. Next, UML is more abstract and therefore introduces new concepts that do not exist in the implementation language. Further, developers may be mislead into believing that UML is a simple because it is a graphical notation.

Last, the scarcity of information on this topic shows that not enough attention has been devoted to this. Future studies that are able to should use effort on enriching this area by actively measuring the necessary learning effort and investigating the kind of that training is necessary to become proficient with UML.

4.3.6 RQ5: How is UML being used in the industry?

The topic of how UML is being used in industry is looked at from both an organizational level and a language-oriented level. The organizational view looks at the background of the UML users, the domain that UML is used in, the cost and size of systems it is used on, the main objective for using UML, consistency of UML's use in the organization, and management's level of support for the technology. Next, a detailed view is taken with respect to the parts of UML that are most used, the methodology it is used with, the most popular UML tools, the implementation languages it is used with, and other activities.

A highly related paper to this section is the industry survey on UML [52]. The structure of this section takes this into account and throughout the section it is simply referred to as the *industry survey*. The other relevant papers (experience reports and case studies) in this section are referred to as *the papers* and consist of: the case study on the adoption of UML in legacy development [54], the case study at a company that moved to an OO/UML paradigm [55], the case study on the impact of the OO/UML on the maintainability of real-world mission-critical software [56], Motorola's experience with model-driven engineering [25], and the experience report on use case authoring challenges during maintenance [58].

The industry survey revealed that most of UML's users were highly educated and experienced with OO: 87% had a bachelor's degree and almost half had a master's degree, 57% had over six years of experience using OO while less than 5% had one year or less. They also filled a variety of roles: one third of the users identified themselves as software developers, a fifth as system analysts, and 15% as IT manager or supervisor. Over 75% previously completed at least one project in UML, helping to give credibility to the respondents regarding their opinion on the subject.

The domains that it is used in is also very diverse (less than 40% of the respondents identified their domain as IT). This is also reflected by the variety of domains reported in the papers, including IT [55, 58], safety-critical [54], mission-critical [56], and telecom [25]. UML appears to be used on projects of all sizes: a third cost $150k or less, a fifth cost between $150k and $1 million, and a third over a million. Size of the systems was not

covered by the industry survey, but the papers in this review suggest both small [58] and large systems [25, 54]. The size of the teams that used UML was not investigated in the industry survey and unclearly reported in the papers. Having said this, the papers point to team sizes ranging from small (under 10) [55, 56] to large [25, 54].

The main objectives of the respondents for using UML were to capture and communicate requirements (reported by more than half of the respondents), guide development of code (reported by a third), and reverse engineering (reported by 10%). This information was underspecified by the papers, though, in [25] it was to reduce development costs in spite of increasing system complexity and in [54] it was mandatory by regulation due to safety constraints dictating use of a semi-formal development method based on UML, while in [55, 56] UML was included with the adoption of OO technology.

In terms of consistency of UML's use, the industry survey found that it is used on all projects by a quarter of the respondents, by one sixth only on large projects, though almost half of the respondents reported sporadic use. Consistency of use is indirectly discussed in [25], where the trend and goal is towards consistent use.

Last, management's involvement, an important ingredient to the successful adoption of any non-trivial technology, was investigated by the industry survey but not addressed by the papers. A third specified that management fully endorsed UML, a third moderately endorsed UML but limited amounts spent on UML, and a third did not care as long as project was completed on time.

The industry survey investigated UML version 1.X, this also seemed to be the version used by all but one of the papers [25], where UML 2.0 was also used. The industry survey reveals that the three most used diagram types (used by 90% of the respondents) were use case, class, and sequence diagrams, while the remaining diagram types were used by half of the respondents. This information was not consistently reported in the papers; only use case diagrams were explicitly mentioned in all the papers. In the two papers [54, 56] that did specify diagram type, the used diagram types were use case, sequence, class, and activity. Additionally [54] also reports the use of the Statechart diagram. The most common tools in the industry survey were found to be Rational Rose (used by 70%) and Sparx Enterprise Architect (used by 55%). Particulars on the tools used was underreported in the papers; there was only one explicit reference to a tool [54]: Rational Rose. The Motorola experience report [25] indicates use of both commercial and in-house tools, but

does not give details on the commercial tools used. Though the programming languages were not covered by the industry survey, the languages mentioned in the papers were C [25, 54], C++ [25, 54-56], and Java [25, 58]. This is not surprising as these are the most common programming languages (note that the use of C points to UML being used in legacy systems).

Most of the respondents used the *unified process* [82] (30%) as their methodology, while *structured*, *agile*, and *none* represented about 16% each (the remainder of the respondents specified that they used an *other* process). In the papers, two instances of the unified process [56, 58] and two used the waterfall model [25, 54] were reported. This data suggests that UML is in fact method agnostic as it is used by variety of development processes in industry.

Advanced uses of UML were reported in three papers:

- In [54] UML was also used for reverse engineering and testing: use cases were used as input to functional test specifications, sequence diagrams were input to integration testing, and class diagrams were used as input to unit testing.
- In [56] models were used during maintenance for impact analysis.
- The most advanced uses of models appear to be in [25] where the "The degree of modeling maturity has evolved from informal "whiteboard" modeling to formal modeling with simulation to code generation to test-case reuse and automated marshaling code generation."

In summary, the combined results of the industry survey and the papers show that UML is used by highly educated and experienced software professionals working in various technical roles and domains, on projects of various cost and size, by small and large teams. While the most common primary objectives for UML's use are to communicate requirements and to help guide development of code, more advanced goals are also reported like test and code generation. The industry survey shows that most use UML sporadically, though a respectable 25% always use it. Only a third of the respondents specified that UML has management's full endorsement, this is a problem as the technology is too complex for large gains to be made by passive management practices.

Not surprisingly, the most commonly used diagram types are use case, class, and sequence and the most commonly used tool is Rational Rose (probably due to historical reasons).

What is interesting is the fact that results clearly show that UML is methodology agnostic and is used in industry for advanced purposes like code and test generation.

In future industry surveys it would be interesting to also have data points on size of projects that UML is used on in terms of different metrics (e.g. LOC, models, classes, use cases), team size, maturity in the manner that UML is used [4], and advanced uses (e.g. code and test generation).

Last, it is important that future industry reports and case studies that involve UML give enough detail on its use to allow for a better understanding of how UML is used. Examples of information that would be helpful is diagram types used, tools used, the main objectives for using UML, the development phase in which UML was used, and the consistency of UML's use in the company.

4.3.7 RQ6: What are the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry?

This section takes a looks at the reported problems in adopting and using UML in industry with the hope that solving these problems will help ease the spread of UML. Six relevant papers have been identified, all based on industrial experience: the case study on the adoption of UML in legacy development [54], the case study at a company that moved to an OO/UML paradigm [55], the case study on the impact of the OO/UML on the maintainability of real-world mission-critical software [56], Motorola's experience with model-driven engineering [25], the experience report on use case authoring challenges during maintenance [58], and the industry survey on UML [52]. All reported issues fall into one of the following categories: demand for better process, demand for tool improvements, and reported problems targeted at the UML language itself.

The manner in which the papers contribute to these areas is summarized in Table 4.22. The industry survey [52] is handled uniquely: since this chapter only looks at problems areas in UML; in terms of the survey we defined these as survey items where at least 40% of the respondents were critical. Also, as the survey did not investigate opinions on tools, this is denoted by the use of "Not Applicable (N/A)".

Table 4.22: Identified Areas for Improvement

	Process	Tools	Core UML
[54]	X	X	X
[55]	X		
[56]	X		X
[25]	X	X	X
[58]	X	X	
[52]	X	N/A	X

Process

Problems with the process in terms of the manner in which UML is adopted and used, is discussed in all the papers. In terms of adoption the authors in [55] note that the adoption of the new technology, which included UML, involved far more than buying a tool and giving some training. Since the technology causes a large change to current working practices, adopters need to invest time and effort into reorganizing their entire development process, creating a new development infrastructure, and giving the staff serious re-training in the new environment. Other adopters need to take these issues into account and budget for this in order to maximize the probability of success.

In the Motorola paper [25] it is reported that the major obstacles encountered by many teams during adoption were due to the lack of a well defined model-driven engineering process, missing skill sets, and inflexibility in changing the existing culture. A missing process leads to "trial and error" approach where the time is spent on the same set of pitfalls others have already experienced. The authors stipulate the need for training in formal languages, modeling, simulation, tools, code generation, model performance improvement, testing automation, and proper partitioning of architectural and design views.

The problem of UML adoption in legacy system development is discussed in two papers [54] and [25]. There exists no well-defined process for the use of UML in legacy system development. This is significant as a large proportion of software development work in industry is enhancement of existing systems. Thus UML must frequently be introduced in legacy development, to leverage previous, often significant investments [83].

In terms problems of working with UML after adoption, ambiguity is flagged by [52] and [25]: 62% of survey respondents felt that "some UML models are insufficiently specified, allowing developers to interpret them in more than one way" [52] and "during the construction of models, implicit assumptions are also made about domain semantics" [25].

The industry survey [52] also found serious difficulties with consistency when comparing or aggregating data from two or more different UML diagrams (experienced by 54% of the respondents). This problem was exacerbated by the fact that concepts are often defined differently in different models (experienced by 44% of the respondents).

Next, the experience report in [58] discusses the issue of managing use case change during system evolution; the article stemmed from a struggle with this issue and a solution based on use case deltas is proposed.

Last, model scalability issues were discussed in [25]. Large systems are developed by separately managed teams that can also be geographically distributed. Model-driven engineering adds another dimension to such a distributed development, since, in order to take advantage of model-driven engineering at the whole system level, one needs to build the subsystem interfaces at the model level (not just at the target level). Further, modeling such a large system becomes very complicated due to information overload, mainly because there are no straightforward ways of hiding detailed information at the right place in design, simulation, and presentation.

Tools

Next is the issue of tools and tools are very important to the success of a method (see Section 4.3.4). Without the existence of good tools supporting the necessary functionality, developers are less likely to use them. Three of the papers of the five papers discussed this issue: in [54], the authors note the need for tool support for reverse engineering code to UML models on legacy systems (see Section 4.3.3.4). Next, need for tool support for managing use case "deltas" (see Section 4.3.3.5) is pointed out by [58]. Finally, problems reported in the Motorola paper [25] are: the exact meaning of a model often is dependent on the tool in which it was created, a lack of UML tool interoperability and poor tool performance (see Section 4.3.4).

Core UML

Last of the reported issues dealt with the UML language itself where four out of six papers reported problem areas. First, sequence diagrams were criticized in two papers; in [54] the authors note that some developers felt that the syntax of UML 1.3 lacked expressive power, with respect to, for example, loops and guard conditions. Next, in [56] it was found that sequence diagrams showed too many repetitive operations and were therefore of little value.

The lack of well defined semantics was identified as a problem area by both the industry survey in [52] and the Motorola case study [25]. Half of the respondents felt that UML semantics is not sufficiently precise to capture some important analysis and design concepts. In the Motorola case study reported that even UML 2.0 contains semantic variation points.

In [56] the authors also commented that UML was regarded as insufficient for expressing all business details (business details were documented in operation specifications). Though from the paper it is not clear what the specific problems were. Similarly, the industry survey [52] found that 56% of the respondents felt that UML is missing critical constructs that would be useful in performing analysis.

Finally, 43% of the survey respondents felt that it was not always clear how to use UML's notations across the different diagram types.

Discussion

Problems experienced during the adoption of UML show the need for processes for both a full and a graduation adoption of UML into a system. In the case of a legacy system, the process must be further specialized. Further, a successful adoption process hinges on serious training and staff adjustment all the way up to the management level. Problem experienced after UML's adoption show that process must also be designed to ensure effective use of UML after its adoption. Areas of concern are model uniformity and ambiguity, i.e. model consistency (between models) with respect to semantics. Model reviews and constraints help greatly with this issue (see Section 4.3.3.2). Next is the issue of ensuring support for system evolution activities such as merging of models and generally dealing with change deltas. Last, the process must be also take into account large system development with models: dealing with a large number of interconnected models, development of models with geographically distributed teams. These are all issues that must be addressed so that UML can be use efficiently and effectively on complex projects.

Next, reported problems with the tools show a certain maturity in the way UML is being used as the requests target features that allow advanced uses of UML: support for using UML with a legacy systems, use case evolution, efficiency of working with a large number of models, and tool interoperability. In terms of the last point, OMG has a direct effort for tool interoperability with the UML 2.0 Diagram Interchange Specification [84].

Last, various issues were targeted at the language itself. Sequence diagrams were criticized in two papers; unfortunately it was unclear whether these problems were due to a lack of training or problems with tools. Next, a lack of well defined semantics was criticized. This is not surprising, the UML specification is large and changes frequently – there are bound to be inconsistencies; having said that, it is clear that UML is stabilizing and becoming increasingly more precise.

UML was also sometimes regarded as being insufficient for expressing all business details. Again, there is no detail discussing exactly what is meant by this. It is entirely possible that these issues can be addressed with the use of OCL, stereotypes and training. Also, it was not always clear to developers how to use UML's notations across the different diagram types. This issue can probably be addressed with training (see Section 4.3.5).

4.4 Threats to Validity

The main threats to the validity of our review we have identified are these:

Choice of Journals and Conference Proceedings / Grey Literature: The review considers a limited number of conferences and journals and omits grey literature. Due to the technique employed for identifying the relevant papers (see Section 4.2.2) judgment had to be used in selecting the sources that would be included to make the work fit a realistic timeframe. Further, unlike peer reviewed publications, grey literature may not have passed any forms quality control.

Unfortunately, the drawback of systematic reviews is that relevant papers may not be identified because they do not conform to the search criteria, a price that is paid for the advantages of a systematic review: completeness (lack of selection bias) and repeatability.

Unpublished Results: In fact, no matter how many more sources would be included, the problem of incompleteness would remain as not all results are published due to various reasons: non-significant results may not be accepted for publication due to the peer-review process, company may choose not to publish negative results (e.g. on failed projects).

Data Extraction & Data Analysis: When extracting data from papers there is a certain degree of subjectivity in terms of what is and is not determined to be relative. Further, bias can affect the interpretation of the results.

4.5 Summary

This chapter presented a systematic review on the effects of UML during the maintenance of OO software, limited to studies that involved developers. The review was systematic so that it could be performed in an unbiased and repeatable manner. Another benefit of systematic reviews is that they can be continued in the future to gauge how the knowledge on the topic has grown.

Thirteen top journals and conferences were selected, of these 1572 papers were found to have used the UML keyword. These were then scrutinized, yielding 23 relevant papers. These papers were used to answer six research questions investigating experiences with UML in terms of costs and benefits, effective use, tool support, learning curve, industrial use, and reported issues. All but one of the relevant papers contributed to the first two research questions, nine also contributed to the remaining four research questions. Most of the studies consisted of experiments, though there were also three case studies, two experience reports, and one survey.

Six papers contributed to the understanding of the costs, risks, and benefits of using UML, these revealed that UML can be beneficial in terms of communication and documentation, correctness, testing, design, and effort. Costs were reported to be training of staff, purchase and integration of tools, and construction of the diagrams. The risks included misinterpretations of inconsistent and incorrect models.

Most of the relevant papers contributed to effective ways of using UML (74%). Due to UML being a large standard with many facets, this research question was divided into several more specific areas. The most tangible results include the effective employment of constraints (stereotypes, modeling conventions, UML to code mappings) and use case points method for effort estimation.

There was surprisingly little information on experiences of working with commercial UML tools (discussed in four papers). Criticism of the tools centered on lack of model-consistency checking functionality, poor model/code synchronization, and poor tool interoperability.

The least addressed area was the UML learning curve; only three papers indirectly broached the topic. While it may be easy to be mislead into thinking that because UML is a visual notation it is therefore easy, the evidence consistently showed that UML is not trivial to learn.

UML appears to be used in various technical roles and domains, on projects of various cost and size, by small and large teams. While the most common primary objectives for UML's use are to communicate requirements and to help guide development of code, more advanced goals are also reported like test and code generation.

For UML to be successfully implemented and widely adopted in industry it needs to be supported by adequate processes and tools. The process must take into account that adopting UML is a non-trivial undertaking. Staff must have proper training, tools, and commitment from management. Tailored processes must be used for UML adoption in new system and legacy systems, where adoption poses additional challenges. Last, tools must be able to handle advanced tasks like handling a large number of models and use case evolution.

One of the goals of this survey was to direct future research. An important area receiving very little attention is diagram use, specifically investigating the type of information, amount of information, and the way in which the diagrams should be used together in order to maximize UML's efficiency and effectiveness. Next is the process for adopting UML into new and existing environments. Also, more papers from industry are needed to shed light on the type of problems practitioners actually face. These papers must not omit information pertaining to the UML tool use and the types of diagrams that were used and for what purposes. Last, research on UML tools should focus on advanced topics like code synchronization and reverse engineering.

5 A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance

5.1 Introduction

This chapter attempts to evaluate the costs and benefits of using UML through a controlled experiment performed at the Simula Research Laboratory in Oslo, Norway. The primary strength of this experiment is the level of realism: it involved 20 professional developers (intermediate to senior level consultants) individually performing the same five maintenance tasks to the same real, non-trivial system where ten of the developers worked with a UML-supported development environment and UML documentation, whereas the other ten developers used the same tools but had no UML documentation to read or update. The developers took one to two weeks to implement the change-tasks. An additional objective was to identify reasons for varying results and therefore identify plausible and necessary conditions for UML to be effective. Our decision to answer the above research question with a controlled experiment stems from the many confounding factors that could blur the results in an industrial context. The experiment fills a gap identified in the systematic review: the inexistence of an experimenting with professionals directly measuring the costs and benefits of UML with a high degree of realism. Further, this experiment builds on expertise acquired in the first two experiments of this type [57], with this one being the first experiment asking these questions in such a realistic setting. A detailed, structured comparison of this and the previous work is presented in Section 5.5.2. Note that this chapter is an extended version of the paper that is published in IEEE Transaction on Software Engineering [7].

The remainder of this chapter is structured as follows. Section 5.2 reports on the planning of the controlled experiment and results are presented in Section 5.3. Section 5.4 analyzes the threats to validity. Related work is discussed in context in Section 5.5. Improvements to the UML tool used in the experiment are suggested in Section 5.6 and conclusions are drawn in Section 5.7.

5.2 Experiment Planning

This section reports on how the experiment was designed and conducted.

5.2.1 Experiment Definition

We wanted to analyze the effects of UML for the purpose of evaluating the costs and benefits of modeling artifacts with respect to effort, functional program correctness, and the design quality of the solution. An important aspect was to decide what the baseline should be against which to compare the use of UML. There are, of course, an infinite number of possibilities here, given the wide variation in software development practices. However, in our experience, the most common situation can be defined as follows: 1) source code is the main artifact used to understand a system, 2) source code is commented to define the meaning of the most complex methods and variables, and 3) there exists a high-level textual description of the system objectives and functionality. This situation is, therefore, what we will use as a basis of comparison in order to determine whether the abstract representations captured by UML help developers to perform their change tasks.

The experiment attempted to answer the following research questions:

(a) Does the provision of UML documentation reduce the effort required in correctly implementing the change tasks?

(b) Does the provision of UML documentation increase the functional correctness of the delivered solution? Since a fault found after the release of the software is significantly more expensive to fix than one found during development [5, 6], special attention must be paid to whether UML increases the probability of the change being functionally correct.

(c) Does the provision of UML documentation improve the design quality of the delivered solution? Alternatively, does the use of UML decrease the decay of a system's design caused by maintenance tasks?

(d) What are the shortcomings of the used state-of-the-art UML tool and how can it be improved?

5.2.2 Context Selection

The context selection is representative of situations where professional Java programmers perform realistic maintenance tasks for the duration of one to two weeks on a real, non-trivial system. Furthermore, the system is initially unknown to the programmer and we are

thus in the common situation where maintainers are not the initial developers of the system.

More specifically, 20 professional developers were recruited from Norwegian consulting companies and paid the negotiated hourly wages. The advantage of using experienced professional developers is so that we avoid one of the main criticisms of most controlled experiments in software engineering: as opposed to student experiments, our results are representative of developers with industrial skills. Furthermore, unlike industrial case studies which typically also use professional developers, this experiment controls for many extraneous factors that can impact our ability to analyze the effect of UML on software maintenance.

Since the ten subjects working with the UML had various degrees of experience and knowledge of UML, they were all given a one-day refresher course which dealt with UML elements with which familiarity was necessary for the experiment. This time was also used to introduce the subjects to the selected UML-supported IDE: Borland Together for Eclipse (BTE) [85]. BTE was selected as the modeling tool due to (1) the advanced synchronization feature between the model and the code and (2) the tight integration with the Eclipse IDE.

5.2.3 Hypothesis Formulation

Our experiment has one independent variable (the use of UML documentation in a UML-supported IDE) and two treatments (*UML*, *no-UML*). It has six dependent variables on which treatments are compared:

- *T:* time to perform the change *excluding* diagram modifications
- *T'*: time to perform the change *including* diagram modifications
- Functional correctness in terms of:
  - (i) *C*: Number of submissions of a solution with a fault
  - (ii) *C'*: Number of submissions of a solution with a fault where the fault broke existing functionality – a subset of *C*

(iii) *C''*: Number of submissions of a solution with a fault where the fault stemmed from not taking into account all existing behavior (*C''*) – a subset of *C*. An example of a fault of type *C''* would be a scenario where the developer must update two packages to correctly handle some new functionality, but, due to a lack of understanding of the system, only updates one of those packages.

- *Q*: design quality in terms of following proper object-oriented design principles [12]. This was calculated by first breaking each task into *subtasks* and rating each as either *acceptable* or *unacceptable*, according to the pre-defined criteria elaborated upon in Section 5.2.6. *Q* counts the number of acceptable subtask solutions.

Following the example and logic in [57], when comparing the time spent on tasks across UML and no-UML groups, one should, of course, account for the overhead involved in modifying UML diagrams. Bearing this in mind, *T'* is a priori a better measure than *T* when assessing the economic impact of using UML. However, we believe that it is still relevant to assess *T* as such results will provide evidence regarding whether UML, as a minimum requirement, facilitates the understanding and change of code. Furthermore, the time spent on modifying the models probably depends strongly on the modeling tool used and the subject's training in that particular tool. This is highly context-dependent and we therefore wanted to distinguish the time developers spent understanding and modifying the code (with the help of UML diagrams) from the time spent on modifying the UML diagrams. The two measures of time are expected to provide interesting, complementary insights.

Two subsets of faults with respect to functional correctness *C* are examined independently: *C'* and *C''*. *C'* only measures the number of faults that led to existing functionality being broken (as opposed to functionality that was added as part of the task), while *C''* only measures the number of faults that stemmed from the developer not adapting existing functionality to work with the newly added functionality. While both *C'* and *C''* show a lack of understanding of the system being modified, faults of type *C'* may be prevented with the assistance of a complete regression test suite.

The hypotheses for testing the effect of UML documentation on our dependent variables are given in Table 5.1. The alternative hypotheses (*Ha*) state that using UML documents improves five out of the six dependent variables: less time to complete the tasks when *excluding* diagram modifications (*T*), improved correctness in terms of *C*, *C'*, and *C''*, and improved design quality (*Q*). Thus Table 5.1 defines five of the hypotheses as one-tailed, because we expected that using UML documentation would help people understand the system design better, and hence provide better solutions faster. However, it is difficult to have clear expectations regarding the effect of using UML documentation on time when *including* time spent on diagram modifications (*T'*), because the time taken to modify the diagrams might be greater than the expected time gains. Thus, the hypothesis on time

Table 5.1: Tested Hypotheses

Dependent variable	Null hypothesis	Alternative hypothesis
Time excluding diagram modifications	H_0: $T(UML) \geq T(no\text{-}UML)$	H_a: $T(UML) < T(no\text{-}UML)$
Time including diagram modifications	H_0: $T'(UML) = T'(no\text{-}UML)$	H_a: $T'(UML) \neq T'(no\text{-}UML)$
Correctness – Num. of submissions of a solution with a fault	H_0: $C(UML) \geq C(no\text{-}UML)$	H_a: $C(UML) < C(no\text{-}UML)$
Correctness – Introduced a fault breaking existing functionality	H_0: $C'(UML) \geq C'(no\text{-}UML)$	H_a: $C'(UML) < C'(no\text{-}UML)$
Correctness – Introduced a fault stemming from not taking into account all existing behavior	H_0: $C''(UML) \geq C''(no\text{-}UML)$	H_a: $C''(UML) < C''(no\text{-}UML)$
Design Quality	H_0: $Q(UML) \leq Q(no\text{-}UML)$	H_a: $Q(UML) > Q(no\text{-}UML)$

including diagram modifications (*T'* in Table 5.1) is two-tailed.

The hypotheses will be tested on the results of each task that the subjects perform (at the task level), as well as on the aggregated results of all the five tasks (across all the tasks, at the subject level). In the case of design quality, the hypothesis will also be tested on each subtask (defined in Section 5.2.6) of every task. Splitting tasks into subtasks allows for a comparison of the quality of solutions across developers, while, at the same time, allowing for a large degree of freedom in the way they implement their solutions.

5.2.4 Selection of Subjects

Subjects were recruited via a request for consultants being sent to Norwegian consulting companies. The request specified a flexible range of time for which the consultants would be needed along with the required education and expertise. Companies replied with

resumes of potential candidates, these were then screened to verify that they indeed complied with the requirements. The subjects were required to at least have: a bachelor's degree in informatics (or equivalent), some familiarity with UML (use case, class, sequence, and state diagrams); and some project experience with the following technologies: Struts [86], JavaServer Pages (JSP) [87], Java 2 [88], HTML [89], the Eclipse IDE [90], and MySQL [91].

Note that the recruitment of all subjects could not be completed before the start of the experiment. This was due to several practical reasons: (1) the market for these skilled professionals is very tight, (2) we could not give the consulting companies a definite start and end date when the consultant would be working, (3) the consulting companies could not give us an exact start date for consultants, and (4) the consulting companies often could not guarantee that the consultant would be available. Consequently, we assigned the first 10 subject to the no-UML treatment, and the next 10 subjects to the UML treatment. This assignment was also beneficial from a logistical point of view, since, at a given point in time, all subjects followed the same experimental procedures. Though this assignment is clearly not "random", there is no reason to believe that the time at which subjects were available was in any way related to their skills. It was rather determined by extraneous factors (e.g., contract terminations) and we had therefore no reason to expect any bias in the assignment process. This was confirmed by the analysis of Table 5.2, which provides background data on the subjects that participated in the experiment, and which clearly shows that the two groups are indeed comparable in terms of age, education, and experience. Furthermore, simple statistical tests on the data in the table confirmed that none of the differences between the groups are significant.

5.2.5 Experiment Design

The experiment was conducted on the BESTweb system [92]. The BESTweb system is a company-internal web-based system developed in Java using the Struts framework [86], written and documented by the author of this thesis. BESTweb supports research on software cost and effort estimation through the identification of relevant journal and conference papers [49]. The system is a database front-end client that gives access to information about all journal papers on software cost and effort estimation that have been coded according to the classification categories *research topic*, *estimation approach*,

Table 5.2: Descriptive Statistics – Subjects' Background

Variable	Group	Mean	Standard Deviation	Min	Lower Quartile	Median	Upper Quartile	Max
Age	No UML	31.7	5.9	25	28	30	37	44
	UML	34.5	5.2	28	31	32.5	39	45
Degree (1=bachelors, 2=masters)	No UML	1.7	0.5	1	1	2	2	2
	UML	1.7	0.7	0	2	2	2	2
Graduation Year	No UML	1997.7	3.1	1991	1996	1998.5	2000	2001
	UML	1997.6	3.8	1990	1995	1998.5	2001	2002
Years of Study at University	No UML	4.9	1.9	3	3	5	6	9
	UML	4.7	1.2	2.5	4.5	5	5.5	6
Years of Study in Computer Science	No UML	3.7	1.4	2	2.5	3.5	4.5	6.5
	UML	4.0	1.7	1	3	4.75	5	6
Average Grade in Computer Science Courses	No UML	2.1	0.4	1.7	1.8	2	2.5	2.8
	UML	2.2	0.5	1.5	2	2.1	2.5	3
Years Of Experience With (YOEW) Java	No UML	4.9	1.4	2	4	5	6	7
	UML	4.8	2.1	1	3	5	7	8
YOEW Servlets / JSP	No UML	2.8	1.3	0.5	2	3	4	4
	UML	2.9	1.9	0.3	1	3	4	6
YOEW Struts	No UML	0.9	0.5	0.3	0.5	1	1	2
	UML	1.3	1.5	0	0.021	1	3	4
LOC in Java	No UML	187100.0	320433.2	6000	30000	50000	100000	1000000
	UML	255500.0	397649.0	5000	20000	75000	200000	1000000
LOC in C++	No UML	3200.0	6663.3	0	0	0	1000	20000
	UML	7000.0	11595.0	0	0	0	20000	30000
LOC in C	No UML	1300.0	3199.0	0	0	0	0	10000
	UML	1100.0	2079.0	0	0	0	1000	5000
LOC in C#	No UML	0.0	0.0	0	0	0	0	0
	UML	1300.0	3199.0	0	0	0	0	10000
Used Borland Together	No UML	40%	/	/	/	/	/	/
	UML	20%	/	/	/	/	/	/
Used a UML Tool	No UML	90%	/	/	/	/	/	/
	UML	90%	/	/	/	/	/	/

research approach, *study context* and *data set*. Thus, each paper in the system is associated with codes based on this classification scheme; these codes are called the *BEST-codes*. Table 5.3 provides the basic metrics for the BESTweb system.

Table 5.3: BESTweb System Metrics

Number of Packages	6
Number of Classes	50
Lines of Java Code	2921
Number of JavaServer Pages (JSP)	17
Number of Attributes	107
Number of Overridden Methods	33
Number of Methods	275
Total Number of Children	13
Maximum Depth of Inheritance Tree	3
Number of Libraries Used	20

The experiment was conducted in two phases, for reasons explained in Section 5.2.4. The subjects in the first phase worked without the UML environment/artifacts. The subjects in the second phase worked with the UML.

The UML documents provide information at a level of detail that one would expect at the end of the design phase [12], including: a use case diagram, sequence diagrams for each use case, and class diagrams. These correspond to the most commonly used diagrams in practice and we wanted our results to be as realistic as possible. For the same reason, all conditions in sequence diagrams were simply described in English. The subjects that received UML documentation had to keep it up-to-date. Table 5.4 and Table 5.5 provide some basic metrics on BESTweb's sequence and class diagrams, respectively. Note that the largest diagrams in both tables are in bolded text, as we will refer to these in the discussion of the results.

Table 5.4: Sequence Diagrams - Metrics

Use Case	Num. of Objects	Num. of Messages
Login	9	17
Change Display Settings	4	3
Filter Publications List using Selected BEST-codes	10	11
Show All Publications	3	2
User's Search	3	2
Query Search	12	18
Sort Publications List	8	12
View Publication Details	7	7
Prepare for showStatistics.jsp	9	8
View Statistics of Publications Per Year	8	7
View Statistics of Publications Per BEST-code Category	13	16
Add User	7	15
Remove User	11	13
Upload a Library File	10	22
System Initialization	5	4
Load BEST-codes and Publications	**17**	**30**

Table 5.5: Class Diagrams - Metrics

Class Diagram	Num. of Classes
no.simula.bestweb.web	**36**
no.simula.bestweb.web.admin	10
no.simula.bestweb.pubmdl	6
no.simula.bestweb.parser	8
no.simula.bestweb.index	6
no.simula.bestweb.db	8

Also, the no-UML documentation was provided to all the subjects. This documentation included: the user's manual, a high-level description of each package in the system and how it relates to the other packages, the third party libraries, the database schema, and the system deployment instructions. The architecture description document was aimed to reflect the type of document that exists in industry for proprietary systems. (At the debriefing session, all but one of the subjects agreed that this documentation was at least as good as the industry standard, 7 out of 20 subjects thought it was better.) The developers also had access to the Javadoc documentation, with which all the code was thoroughly documented.

In order to maximize the realism of the experiment the subjects were not informed of other participants, furthermore they were told that since this work was being performed for a software research laboratory we wanted to take advantage of the situation to collect data in order to learn how professional software developers work. The subjects were informed in

advance that they would be working on a system that was also involved in a study. We deemed these steps necessary so that the developers would take the work seriously and not treat it as an exercise. Lastly, the consultants signed a non-disclosure agreement in order to make sure that they would not disclose information about their work to a potential future subject.

The subjects were also told that, for us to collect valid data, a few rules had to be followed. First, they needed to use the pre-configured development environment (e.g., the Eclipse IDE). Next, they had to work independently: they could not get help from colleagues or the experimenters. Technical questions to the latter had to be asked via email. The reason for this was twofold: so that the subjects would not engage in a technical conversation with the experimenters and so that answers were carefully considered by the experimenters (to not give an unfair advantage to the subject). Furthermore, only one task would be given out at a time and the total number of tasks would not be disclosed: this ensured that the developers avoided budgeting their time. Finally, the introductory and debriefing sessions were to be audio-recorded.

The subjects went through the following procedure, illustrated in Figure 1:

1. The subject is given an introductory session explaining the manner in which he would be working.
2. The subject answers an initial questionnaire capturing the subject's background and experience – see Table 5.2.
3. If the subject is in the UML group; the subject receives a UML refresher / tool training session.
4. The subject receives the first task.
5. The subject submits an estimate of the amount of time that they think the task will take them.
6. The subject implements the task.

7. Upon completion, the task is sent in for acceptance testing. The system is then tested by an experimenter based on a system acceptance test plan.
    a. If the test fails, the subject is told the problem and asked to fix it to submit the solution again.
    b. If the test passes, the subject receives the next task and repeats the process from Step 4.
8. Upon completion of all five tasks: Debriefing.

Since the experiment was conducted in a manner that was very labor-intensive for the experimenters, a maximum of three subjects participated in the experiment at a time. Essentially, the author of this thesis acted as an advanced "customer" that received the intermediate results described above from the subjects, tested and accepted solutions, and in general ensured that they did not deviate from the prescribed process.

In the case of the tasks where the developer modified existing functionality, the test plan was derived from the use case diagram and the sequence diagrams [93]. It ensured that, for every task, all relevant functionality was tested by covering all messages in every relevant sequence diagram. Exceptional scenarios and conditional flows were also accounted for (this sometimes resulted in the same path being covered several times). If the task required the addition of new functionality to the system (i.e. the addition of a new use case), checklist-based testing was applied [94]. The checklist was derived from the functional specifications and checked: the main flow, exceptional scenarios, and existing functionality.

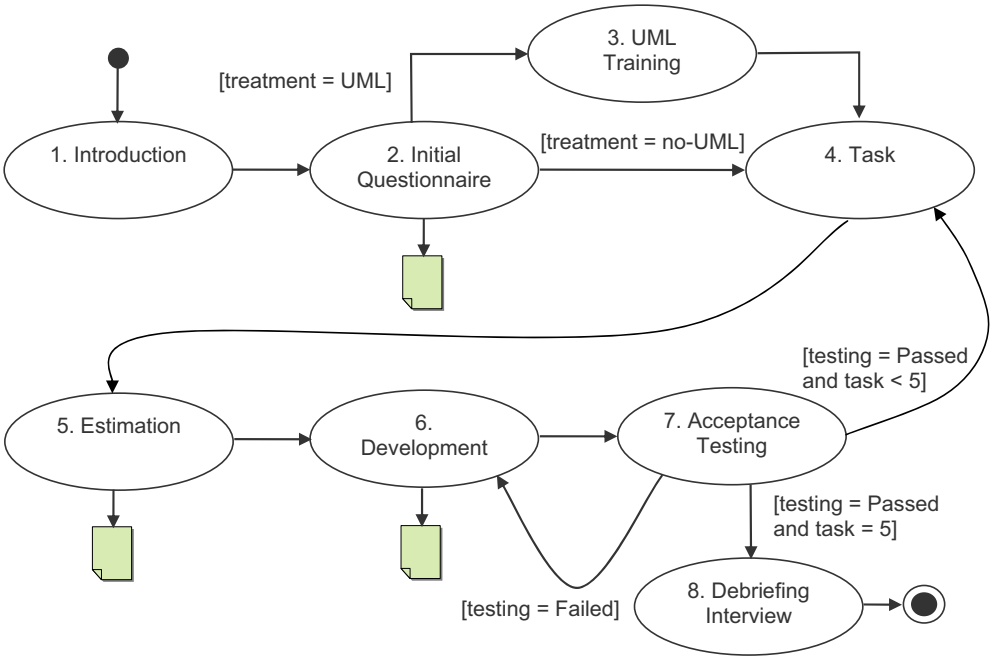


Figure 1. The experimental process (the documents represent deliverables)

5.2.6 Instrumentation and Measurement

The instrumentation and measurement process was specified before the experiment began, and outlined exactly how the interaction with the subjects would be performed; it also outlined how the data would be collected when interacting with the subjects.

The data sources were (see Figure 1):

- For every subject, an initial questionnaire capturing the subject's background and experience – see Table 5.2.
- For every submission, a copy of their entire source code.
- For every submission by a UML subject, the estimated percentage of time they spend on reading and updating the UML.
- For every submission, acceptance test reports (generated by the experimenters).
- For every subject, an audio-recorded semi-structured debriefing interview (conducted after the developer finished all their tasks), see Section 5.2.8.2.

An important point in this experiment was the fact that a subject's solution (submission) was not accepted until it passed all the functional tests. Such a setup ensures that (1) in their ultimate form, all tasks conform to the predefined specifications, and (2) every subject completes every task. Full conformance to specifications is important to be able to compare final solutions in terms of effort, for example. Some subjects being slower than others, it was important not to fix the time allocated for tasks to ensure point (2) and to be able to observe differences in effort. The disadvantages of this decision are related to cost and logistics. Cost is higher as the slower subjects will need to be paid more to complete the tasks. The logistics are more difficult as it cannot be anticipated how long the subjects will take to complete the tasks.

The number of resubmissions and the reasons for their need were recorded. The resubmission problems were categorized as either *omissions* or *faults*, where a submission that did not fully implement the specified functionality was defined as an *omission*.

The solutions for each task were also assessed for their design quality in terms of following proper object-oriented design principles [12]. This was done by first breaking each task into *subtasks* and then specifying all acceptable solutions for that subtask. Each *subtask* corresponds to a subset of an entire task's functionality; the level of granularity was set to one high-enough to make it possible to compare the corresponding solutions (code) across all the subjects. For example, in the case of a task which requires access to the database, the code that accesses the database could be placed almost anywhere, even though the proper place for it is in the package that specifically deals with the database interaction. Even though code that is placed in, say, the presentation layer, would pass the functional-correctness tests, this would be an unacceptable solution (leading to code decay). Thus, each possible solution for each subtask was rated as either *acceptable* or *unacceptable*, according to the pre-defined criteria. Specifically, a solution to a subtask with one of the following problems would be deemed as *unacceptable*:

- Duplication (copying and pasting) of existing code instead of direct use of that logic (e.g., copying and pasting a sort method).
- Addition of new design elements that current design elements could have handled (e.g., creating a new partial User class even though an existing User class is present).
- Incorrect placement of logic in a class.

- Distributing logic throughout the application when adding it to just one place would have had the same effect.
- Use of the try/catch mechanism as a normal part of the application's logic.

5.2.7 The Tasks

In total, the subjects performed five maintenance tasks in a fixed order. It was estimated that it would take the subjects between three and six days to complete all the tasks (assuming 7.5 hours/day; not including non-development activities). The tasks consist of adding necessary features of practical value and affect every part of the system: at least one class in every package has to be modified. A summary of necessary modifications is given in Table 5.6.

5.2.7.1 Task 1

The first task requires the addition of functionality to save a user's search-query to persistent memory. Also, the user's last search-query must be read and re-executed automatically upon their re-logon to the system.

The task requires that the developer understands how the system starts-up, interacts with the database, and executes a query. In terms of coding, the task is small in size, as one of its purposes is to introduce the subject to the system. However, initially, the task may seem complex because developers must gain a basic understanding of several parts of the system before the task can be solved.

Table 5.6: Minimal modifications necessary for implementing each task

Task	Necessary Modifications
1	At a minimum, the task requires modifications to 3 classes containing respectively 5, 10, and 80 lines of code.
2	The task is unique in that it has only one correct solution which the developer can obtain in a limited and repetitive manner. The task requires a modification of a few lines of code in nine classes.
3	At a minimum, the task requires the addition of six new classes, two JSPs, and the modification of one class and one JSP.
4	At a minimum, the task requires a modification of four classes (30 LOC in total) and the addition of three other classes (80 LOC in total).
5	At a minimum, the task requires the addition of four classes and two JSPs, and the modification of one class and one JSP.

5.2.7.2 Task 2

The second task requires the extension of the system to handle an additional piece of data from an input file (in XML format) used to update the publications in the BESTweb system. The system already partially handles the data: if it encounters the presence of the data in the file it warns the user that this data is not supported. The developer is asked to add support for this data by extending: the domain model, the GUI and the search functionality.

The task requires that the developers gain an understanding with the system's domain model, GUI, and search functionality. The task can also be solved without a complete understanding of the implications of their changes by searching for every occurrence of a similar piece of data, and then copying/pasting and modifying the code accordingly. For example, if the domain model has a *publication* and the system already handles the *title* property of the *publication*, but not the *author* property, the developer could search for every occurrence of *title* in the system, then copy and paste the code that handles *title* and replace *title* with *author*.

5.2.7.3 Task 3

The third task requires the developer to add completely new functionality to the system (as opposed to making modifications to existing functionality). Recall that the BESTweb system was designed to aid in working with cost and effort estimation papers [49]. Each one of these papers is classified according to a categorization scheme (using the BEST-codes). This task asks the developer to add functionality to the system that extends the manner in which cost and effort estimation metadata associated with each publication is dealt with, specifically, the ability to add categorization categories and corresponding codes. Without such functionality the user would have to manually add the code in the database and restart the system. The restart would be necessary so that the new code would be associated with the corresponding publications.

The task is complex as it requires the developer to understand most parts of the system. Furthermore, the developer has to create a GUI using JSP and Struts (something that a large number of developers ended up struggling with).

5.2.7.4 Task 4

Task 4 asks the developers to add caching logic to the system so that if statistics for all the publications in the system are requested, the cached results are used (so as to decrease the computational load on the system).

The task requires the developers to understand the statistical data and how it is generated by the system. The developers also have to ensure that when, for example, a new code is added to the system (via functionality they added in Task 3), these cached results are updated.

The task forces the developer to deal with the *Best Codes Manager* (BCM) class. The BCM class is noteworthy as developers frequently found it non-trivial to understand due to the following reasons:

- It contains a cached copy of all the BEST-codes found in the database, this requires that the developers understand how the caching strategy works.
- It retrieves objects from a qualified association that points to another qualified association.
- It contains thread-safe logic.

5.2.7.5 Task 5

The last task is in-fact a continuation of Task 3, thus the comments for that task apply to Task 5 as well. Whereas in Task 3 the developers are asked to add functionality where the user could add new types of publications codes to the system, in this task the developers are asked to add functionality where the users could delete existing publication codes from the system.

5.2.8 Analysis Procedure

The analysis procedure included both quantitative and qualitative components. The quantitative data was the main source for testing the hypotheses, whereas the qualitative data was analyzed in an attempt to gain a deeper understanding of the work processes of the subjects, which in turn could potentially offer additional, complementary evidence and to some extent explain the quantitative results.

5.2.8.1 Quantitative Analysis

Univariate analyses of the dependent variables were performed to test the hypotheses, both, individually for each task, and across all tasks. For all dependent variables (*T* , *T', C*, *C'*, *C''* and *Q*) two-sample *t* tests were performed [95]. In addition, to reduce potential threats to the validity of statistical conclusions resulting from violations of the *t* test assumptions, non-parametric Wilcoxon rank sum tests were also performed [95]. Additionally, with respect to design quality (*Q*), Fisher's Exact Test [95] was used to test the difference in *proportion of subjects* with solutions being scored as *acceptable* for each subtask (of each task).

The level of significance for the hypotheses tests were set to $\alpha = 0.05$. However, the reader should bear in mind that we perform multiple tests and, in order to allow for a stricter and more conservative interpretation of the results (e.g., using a Bonferroni procedure or one of its variants [96]), we provide p-values.

Furthermore, it is often useful to know not only whether an experiment has a statistically significant effect, but also the *size* of any observed effects. Thus, for the dependent variables on time and correctness, the effect size was calculated using Cohen's *d,* which is defined as the difference between two means divided by the pooled standard deviation for those means [97]. In our case, we calculated the difference in means between the UML and no-UML group so that a positive value of *d* corresponded to the UML treatment being *beneficial*. To interpret the results, Cohen suggested that $d = 0.2$ is indicative of a *small* effect, $d= 0.5$ a *medium* and $d=0.8$ a *large* effect size.

Given the small number of subjects, we also fitted multivariate Analysis of Covariance (ANCOVA) models for the time and correctness data across all tasks and subjects. The average grade in computer science courses (see Table 5.2) of each subject was included as a *covariate* to adjust for individual differences between the subjects, and *UML* and *Task* (and their interaction) were the independent variables. The use of the covariate, combined with the fact that we had a total of 100 data points for each dependent variable (20 subjects and five tasks) resulted in increased statistical power compared to the less sophisticated univariate analyses. However, since the observations of individual tasks for a given subject are correlated, the ANCOVA assumptions of independent observations would be violated. We thus resorted to a statistical technique known as Generalized Estimating Equations (GEE) [98] to estimate the parameters (i.e., the effect of *Grade*, *UML* and *Task*) of the models. GEE is an extension of Generalized Linear Models, developed specifically to

accommodate data that is correlated within clusters (here being the individuals). The results of the GEEs were entirely consistent with the univariate results and are presented in Appendix A. Still, it implies that the univariate results presented in this paper probably do not suffer from Type II errors as a result of lack of power, since the multivariate analyses of covariance with 100 data points showed consistent results.

5.2.8.2 Qualitative Analysis

A semi-structured debriefing (interview) session was held with each developer immediately upon completion of the tasks. An interview guide with relatively open questions was prepared and all sessions were audio-recorded. The subjects were asked about their style of working (e.g., how they gained an understanding of the system) and the problems they faced (e.g., what they suspect led to the introduction of every fault). The interviews lasted between 50 and 98 minutes for the no-UML subjects (70 minutes on average) and between 91 and 169 minutes for the UML subjects (123 minutes on average). The interviews varied in length due to several reasons: each error had to be discussed and the total number of errors differed across developers, further, some subjects were more talkative than others. The interviews with the subjects in the UML group took longer since, additionally, the usage of UML was thoroughly discussed. The length of this additional discussion varied depending on the extent to which the developer took advantage of the UML and the developer's prior experience with UML. Developers that did not take complete advantage of the UML diagrams could not comment as much as those who did.

The amount of information extracted from the subjects also varied due to other reasons. The developers spent between one to two weeks on the experiment. The longer it took them to complete all tasks, the harder it was for them to remember details of what happened in earlier phases of the experiment. Unfortunately, the developer could not be interviewed after completion of each task as that would influence her (e.g., discussions examining why the developer chose a certain solution and not an alternative might have given the developer a deeper insight into the system). Furthermore, some people can provide insight into their thought-process better than others and others have a very hard time forming an opinion [99].

Content analysis [51], a data reduction technique, was then applied to the audio recordings of the semi-structured interviews in the following manner:

1. An initial set of codes was derived from the interview questionnaire.

2. The interviews were played-back and the subjects' answers and opinions were transcribed and coded. This made the coding traceable.
3. If a subject's opinion or answer was not possible to code with an existing code, a new code was declared.
4. Once this was done for all the interviews, the codes were then reviewed, refined, and finalized.
5. All the interviews were then replayed and recoded with finalized coding scheme.

The coding schema (see the Appendix) encompassed the problems the developers faced, their thoughts on the system and how they worked. Additionally, the no-UML interviews were coded for problems that occurred due to the models not being available. The UML interviews were coded to capture the manner in which the UML diagrams were used. The purpose of the analysis was to better understand how access to UML documentation made a difference.

5.3 Experimental Results

The results from the analysis procedures described in Section 5.2.8 are now shown and dissected. The section first presents and discusses the descriptive statistics and univariate results, the qualitative results and discussion follow. Lastly, the main results from the experiment are summarized in Section 5.3.5.

5.3.1 Descriptive Statistics and Univariate Analysis

Recall that four hypotheses are tested with regards to the effect of UML on (1) the time to perform the change task excluding diagram modification (using the variable *T*), (2) the time to perform the change task including diagram modification (using the variable *T'*), (3) submission correctness (using the variables *C*, *C'*, and *C''*), and (4) design quality (using the variable *Q*). These are examined respectively.

Note that the Wilcoxon test and the *t* test provide consistent results in all but one case (*C* for Task 1 in Table 5.9 gave a p-value of 0.053). Due to the non-normal distribution of the data, the Wilcoxon p-values are used in the discussions. The significant values also appear in bold font in the tables.

5.3.1.1 Time

Table 5.7 shows the descriptive statistics and univariate results for time (in minutes) across all the tasks and for each task, respectively. For each dependent variable (denoted by the Var column), the results are presented for each treatment (Treat): the mean of the subjects in the group, the standard deviation, the smallest data-point (Min), the lower quartile (Lower Quart), the median (Med), the upper quartile (Upper Quart), the largest data-point (Max), the effect size as a percentage difference between the two means, the effect size using Cohen's d, the p-value from the *t* test, and the p-value from the Wilcoxon test.

In terms of time excluding diagram modifications (*T*), the no-UML group spent 1.4% more time in total. The development time is shorter on the first and the two largest Tasks (3 and 5) for the UML group. The variance is also smaller for the UML group on the largest Tasks (3 and 5), similar on Tasks 1 and 4, and larger on Task 2. The minimum is smaller for the

Table 5.7: Descriptive Statistics & Univariate Results – Time

Task	Var	Treat	Mean	Std Dev	Min	Lower Quart	Med	Upper Quart	Max	% diff	Coh-en's d	*t* test	Wilc-oxon
All	T	No UML	2030.3	921.6	950.0	1318.0	1913.0	2759.0	3458	1.4%	0.04	0.47	0.46
		UML	2001.6	602.8	1112.0	1618.8	2074.1	2429.0	2838				
	T'	No UML	2030.3	921.6	950.0	1318.0	1913.0	2759.0	3458	-14.5%	-0.36	0.43	0.35
		UML	2325.4	697.1	1410.0	1855.0	2314.0	2849.0	3610				
1	T	No UML	327.1	145.0	145.0	183.0	345.0	355.0	565	6.8%	0.16	0.37	0.41
		UML	304.7	142.1	154.0	198.5	298.8	355.0	622				
	T'	No UML	327.1	145.0	145.0	183.0	345.0	355.0	565	-16.1%	-0.36	0.44	0.64
		UML	379.7	151.2	194.0	265.0	331.5	480.0	685				
2	T	No UML	234.9	117.8	130.0	140.0	172.0	332.0	445	-23.3%	-0.39	0.81	0.88
		UML	289.7	161.9	173.5	210.0	220.0	315.0	721				
	T'	No UML	234.9	117.8	130.0	140.0	172.0	332.0	445	-24.0%	-0.40	0.39	0.23
		UML	291.2	162.3	180.0	210.0	222.5	315.0	725				
3	T	No UML	681.4	417.9	323.0	410.0	533.5	765.0	1471	9.0%	0.19	0.34	0.29
		UML	620.4	196.3	337.5	479.0	646.0	722.0	894				
	T'	No UML	681.4	417.9	323.0	410.0	533.5	765.0	1471	-11.1%	-0.22	0.62	0.25
		UML	757.3	237.6	435.0	535.0	800.0	910.0	1125				
4	T	No UML	318.0	157.8	135.0	155.0	325.5	477.0	552	-6.0%	-0.13	0.61	0.63
		UML	337.1	132.1	184.0	230.0	303.0	440.0	542				
	T'	No UML	318.0	157.8	135.0	155.0	325.5	477.0	552	-21.7%	-0.45	0.33	0.35
		UML	387.0	151.0	220.0	270.0	342.5	540.0	640				
5	T	No UML	468.9	295.0	197.0	220.0	377.5	639.0	1119	4.1%	0.08	0.43	0.49
		UML	449.8	200.6	140.0	301.0	454.8	550.3	746				
	T'	No UML	468.9	295.0	197.0	220.0	377.5	639.0	1119	-8.8%	-0.17	0.71	0.49
		UML	510.2	191.8	220.0	355.0	535.0	590.0	812				

UML group in the case of the two largest tasks, and larger in the case of the other tasks. The maximum is smaller for the two largest tasks for the UML group, and larger for other

tasks. However, when including the time that the UML group spent on updating the UML documentation (*T'*) then, overall, the no-UML group finished the tasks 14.5% faster than the UML group. On the two largest tasks (3 and 5), the UML overhead is lowest (around 10%, even though there is a lot of UML to update on those tasks). The variance is also smaller for these largest tasks for the UML group, similar on Tasks 1 and 4, and larger on Task 2. The minimum is always smaller and the maximum larger for the no-UML group in the case of the two largest tasks (3 and 5), and otherwise smaller. None of the differences in time are, however, statistically significant at $\alpha = 0.05$. Furthermore, the effect size measure *d* is well below 0.5 for both *T* and *T'*, and as explained in Section 5.2.8.1, this indicates a *small* treatment effect.

In order to gain a deeper understanding of how the UML subjects spent their time, Table 5.8 shows the percentage of time the UML subjects have spent on reading and updating the UML documentation, on each task. Note that unlike the difference between *T* and *T'*, which compares subjects in the two groups, this measure only deals with the subjects in the UML group and the amount of time they self-reported to have spent on UML. On average, 14.8% of time spent on each task was spent reading the UML and 13.2% of time was spent on updating the UML. Note that in Task 2 there was virtually no UML to update due to the fact that the changes consisted of adding attributes and associations on class diagrams, and those are updated automatically. Also, remember that Tasks 3 and 5 were the largest tasks and were fairly similar while Task 4 was the most complex task. Keeping this in mind, the table shows that the subjects spent the most time reading the UML while working on the tasks that dealt with parts of the system that they were not familiar with (Tasks 1 to 4), as the amount of time spent on reading the UML in Task 5, the only task that dealt with functionality the developers were already familiar with, is significantly smaller. In particular, the largest percentage of time used for reading the UML was spent during Tasks 1 and 2, when the subjects were least-experienced with the system. By Task 5, relatively little time is used on reading the UML: only 6.5%. Next, while the time used for updating the UML is dependent on the amount and types of changes introduced to the code, a general trend can be seen when looking at Tasks 1, 3, and 5. Task 1 requires relatively few changes to the UML, but at this time the developers have very little experience at updating the UML using the tool (BTE). While Tasks 3 and 5 require the most changes, the types of changes are very similar. Thus, one can see that the percentage of time the subjects spend on updating the UML tends to decrease as they gain experience with this activity.

Table 5.8: Descriptive Statistics – Percentage of Time Spent on UML

	Task 1	Task 2	Task 3	Task 4	Task 5	Average
Reading	20.9%	21.5%	10.1%	15.1%	6.5%	14.8%
Updating	20.8%	0.6%	17.7%	12.9%	14.6%	13.2%

5.3.1.2 Correctness

Table 5.9 follows the presentation style used by Table 5.7 and introduced in Section 5.3.1.1, but deals with correctness instead of time. In terms of the number of tasks submitted with a fault (*C*), we see that the UML group had 50% to 100% fewer faults in each and every task, and 54.7% fewer faults overall. The variance is smaller on all but the first task (where it is almost equal) for the UML group. The minimum is zero in both groups, across all tasks. The maximum is always smaller for the UML group except for on the first task (where it is equal). The differences are statistically significant in Task 1 with a p-value of 0.04 and across all the tasks with a p-value of 0.03. Overall, there is a *large* positive effect of the UML-treatment on correctness, as indicated by Cohen's *d* being well above 0.8 across all tasks.

The faults that broke existing functionality (*C'*) occurred 0% to 100% less often in the UML group throughout the three tasks where it was observed (1, 2, and 5), and 50% less overall. There were no significant differences at $\alpha = 0.05$.

Last, faults stemming from not taking into account all existing behavior (*C''*) only occurred in Task 1 submissions (where the subjects are completely new to the system). The UML group had 70% fewer faults of this type with a lower variance. The minimum is zero in all the cases, and the maximum is larger for the no-UML group. The difference is significant with a p-value of 0.02. As for *C*, the effect size measure *d* is well above 0.8, indicating a *large* benefit of UML on *C''*.

Table 5.9: Descriptive Statistics & Univariate Results – Correctness

Task	Var	Treat	Mean	Std Dev	Min	Lower Quart	Med	Upper Quart	Max	% diff	Cohen's d	*t* test	Wilc-oxon
All	C	No UML	5.3	2.7	2.0	4.0	4.0	9.0	9.0	54.7%	1.11	**0.0105**	**0.0313**
		UML	2.4	2.5	0.0	0.0	2.0	5.0	6.0				
	C'	No UML	1.2	2.4	0.0	0.0	0.5	1.0	8.0	50.0%	0.34	0.2390	0.4100
		UML	0.6	0.8	0.0	0.0	0.0	1.0	2.0				
	C''	No UML	1.0	0.7	0.0	1.0	1.0	1.0	2.0	70.0%	1.15	**0.0075**	**0.0191**
		UML	0.3	0.5	0.0	0.0	0.0	1.0	1.0				
1	C	No UML	1.6	1.0	0.0	1.0	1.5	2.0	3.0	50.0%	0.76	0.053	**0.040**
		UML	0.8	1.1	0.0	0.0	0.0	2.0	3.0				
	C'	No UML	0.3	0.7	0.0	0.0	0.0	0.0	2.0	0.0%	0.00	0.500	0.639
		UML	0.3	0.7	0.0	0.0	0.0	0.0	2.0				
	C''	No UML	1.0	0.7	0.0	1.0	1.0	1.0	2.0	70.0%	1.15	**0.008**	**0.019**
		UML	0.3	0.5	0.0	0.0	0.0	1.0	1.0				
2	C	No UML	0.1	0.3	0.0	0.0	0.0	0.0	1.0	100.0%	0.47	0.165	0.500
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
	C'	No UML	0.1	0.3	0.0	0.0	0.0	0.0	1.0	100.0%	0.47	0.165	0.500
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
	C''	No UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0	/	/	/	/
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
3	C	No UML	0.5	0.7	0.0	0.0	0.0	1.0	2.0	80.0%	0.74	0.060	0.125
		UML	0.1	0.3	0.0	0.0	0.0	0.0	1.0				
	C'	No UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0	/	/	/	/
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
	C''	No UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0	/	/	/	/
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
4	C	No UML	1.4	1.8	0.0	0.0	1.0	2.0	5.0	50.0%	0.47	0.149	0.186
		UML	0.7	1.1	0.0	0.0	0.0	1.0	3.0				
	C'	No UML	0.8	1.5	0.0	0.0	0.0	1.0	5.0	62.5%	0.45	0.171	0.361
		UML	0.3	0.5	0.0	0.0	0.0	1.0	1.0				
	C''	No UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0	/	/	/	/
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
5	C	No UML	1.7	1.8	0.0	0.0	1.0	3.0	5.0	52.9%	0.62	0.181	0.130
		UML	0.8	1.0	0.0	0.0	0.5	1.0	3.0				
	C'	No UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0	/	/	/	/
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				
	C''	No UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0	/	/	/	/
		UML	0.0	0.0	0.0	0.0	0.0	0.0	0.0				

5.3.1.3 Design Quality

Table 5.10 shows the descriptive statistics and the univariate analysis results for design quality (*Q*). The first column in the table indicates the design quality criteria, which can be one of the following: the number of subjects with an acceptable solution to a subtask (with a maximum of 10, in which case every subject in the group had an acceptable solution to the task), the subtasks aggregated (summed) to the task level, the subtasks aggregated across all five tasks. Remember that in Task 2 there was no flexibility in the implementation, there was essentially only one way in which the task could be solved to obtain a functionally correct change, thus, data for this task is absent from the table.

Overall, the UML group had 7.3% more acceptable solutions. Furthermore, a significant difference was found for Task 1 (overall), where the subjects lack familiarity with system and are changing existing functionality; the UML group's design quality score was 56.2% better with a p-value of 0.0025. Otherwise, the differences in quality are relatively small and not statistically significant.

Table 5.10: Descriptive Statistics & Univariate Results – Design Quality

Criteria	No UML	UML	% diff	Fisher's Exact Test	*t* test	Wilcox
Task 1, Subtask 1, Acceptable	8	10	25.0%	0.2368	/	/
Task 1, Subtask 2, Acceptable	2	6	200.0%	0.0849	/	/
Task 1, Subtask 3, Acceptable	6	9	50.0%	0.1517	/	/
Task 1, Total, Acceptable	16	25	56.2%	/	**0.0006**	**0.0025**
Task 3, Subtask 1, Acceptable	7	8	14.3%	0.5000	/	/
Task 3, Subtask 2, Acceptable	9	10	11.1%	0.5000	/	/
Task 3, Subtask 3, Acceptable	10	9	-10.0%	1.0000	/	/
Task 3, Subtask 4, Acceptable	7	6	-14.3%	0.8251	/	/
Task 3, Subtask 5, Acceptable	10	9	-10.0%	1.0000	/	/
Task 3, Total, Acceptable	43	42	-2.3%	/	0.6077	0.5577
Task 4, Subtask 1, Acceptable	8	7	-12.5%	0.8483	/	/
Task 4, Subtask 2, Acceptable	9	9	0.0%	0.7632	/	/
Task 4, Subtask 3, Acceptable	6	3	-50.0%	0.9651	/	/
Task 4, Subtask 4, Acceptable	1	2	100.0%	0.5000	/	/
Task 4, Total, Acceptable	24	21	-12.5%	/	0.7268	0.7329
Task 5, Subtask 1, Acceptable	7	9	28.6%	0.2910	/	/
Task 5, Subtask 2, Acceptable	6	6	0.0%	0.6750	/	/
Task 5, Total, Acceptable	13	15	15.4%	/	0.2837	0.3707
All Tasks, Acceptable	96	103	7.3%	/	0.2288	0.2187

5.3.2 Discussion of Quantitative Results

Overall, when looking at the total time (*T*) the subjects spent on the five tasks, we see that the UML group completed the tasks slightly faster (1.4%) than the no-UML group (Table 5.7). This difference is not practically or statistically significant. When we take the time it takes to update the UML documentation into account, we see that the UML group spent 14.5% more time on the five tasks, though this difference is not statistically significant either (Table 5.7). The observed cost of keeping the UML documentation updated can be better understood if we look at Tasks 1 and 5. During Task 1, the UML subjects have very little experience using the UML tool, thus, a certain learning curve can be expected. On the other hand, by the time developers get to Task 5 the UML subjects are fairly comfortable at using the tool and are adding functionality to a system they are fairly familiar with. This is

clearly shown in Table 5.8: during Task 1, 20.9% of the time is spent reading the UML and 20.8% of the time is spent on updating it. By Task 5, a larger percentage of time is spent on maintaining UML (14.6%) than reading it (6.5%). In Task 1, the UML subjects spent 16.1% more time on the task than the no-UML subjects updating the UML documentation (Table 5.7). This goes down by almost half, in Task 5, to 8.8%. Thus, the time spent on UML in Task 1 could be considered a worst-case scenario that can be expected in terms of time overhead. We use the term "worst-case scenario" since this is what the cost of UML would be if there would be no other noticeable gains from the use of UML. But, this is not the case since the UML group performed much better in terms of correctness on every task, 50% to 100% better. The difference is statistically significant in the first task, and across all the tasks (for *C* and *C''*). This makes sense as, during the first task, the developers were least familiar with the system; thus, they were more likely to introduce a fault. This explains why the UML clearly helped the developers on the first task. Taking a closer look at correctness reveals that the UML group always did as well, or better than the no-UML group, even if not all the differences are statistically significant. This is probably due to the fact that the developers in the UML group gained a deeper understanding of the system, thanks to the UML documentation, as further suggested by the qualitative analysis in the next section.

Furthermore, in terms of design quality (*Q*), significant gains are found in Task 1. Our explanation is that, the fact that the developers are completely new to the system is offset by the presence of the UML documentation, significantly so. The subjects in the UML group delivered a higher-quality solution with lower complexity than did the subjects in the no-UML group. This is important as this result suggests that UML can help prevent code decay [10] in real systems when developers are not familiar with the system.

Tasks 2 and 4 did not follow the general trend where *T* was lower for the UML group (and, in fact, *T'* ended up being higher as well). We speculate that the reasons for this are task specific. Task 2 is unique in that it can be solved in a limited and repetitive manner. As discussed in 5.2.7.2, this was possible as all the developer had to do was to extend functionality in the system by the copying/pasting of code and changing the variable name. Furthermore, once the changes were made, the application either worked or it did not. Hence, the developer could have solved the task by having a hunch as to how it can be done, doing the change in a mechanical manner (without fully understanding the consequences of the change), and testing the application to see if the hunch was correct.

The no-UML subjects were more predisposed to solving the task in such a manner as they dealt directly with the code. Once they identified a relevant piece of functionality, it was natural for them to perform a search in the code, instantly revealing the other relevant locations. The UML subjects were less likely to perform such a search as they primarily used the diagrams, and therefore performing a search to find relevant occurrences is not as easy, natural, or possible to perform. Instead, the UML subjects tried to find the right place in the code by studying the diagrams. This procedure inhibited the use of the search/copy and modify approach by the UML subjects. Conversely, this made the UML subjects less likely to guess at the answer. Unfortunately, no quantitative data can backup this hypothesis, apart from the large amount of time the UML subjects spent on this task (even though virtually no UML had to be updated) (Table 5.8).

In the case of Task 4, *T* is slightly higher for the UML group as well; this may be due to the fact that it was the most complex task to solve. The complexity stemmed from the fact that the most complex parts of the system had to be understood before the task could be solved correctly. Thus, if proper due diligence was not performed on the task, faults could easily be introduced.

In summary, the quantitative results show that,

- UML was always beneficial in terms of functional correctness (introducing fewer faults into the software),
- UML was slightly more costly in terms of time if the UML documentation was to be updated (though, slightly less costly if it was not) – though these results were not significant, and
- UML helped produce code of better quality when the developers were not yet familiar with the system.
- The largest gains were experienced during the first task. This is an important finding as developers in industry are often faced with the "first task" scenario due to: high staff turnover and involvement on a very large system (where any one developer is only familiar with a small portion of the system).

Thus, one can conclude that, overall, using UML can be beneficial when a developer must extend a non-trivial system they are unfamiliar with; a very typical occurrence in industry.

5.3.3 Qualitative Analysis Results

First, results applicable to all the subjects are presented and summarized in Table 5.11. Next, results applicable to the subjects in the no-UML group are presented and summarized in Table 5.12. Finally, results applicable to the subjects in the UML group are presented and summarized in Table 5.13. Note that a more detailed breakdown of the data presented in these tables can be found in Appendix B.

Table 5.11 consists of two types of information: rows *a* to *f* reflect the subjects' answers to questions that appeared on the debriefing questionnaire. Rows *g* to *k* reflect information offered by the subjects during the debriefing session (e.g., while discussing the difficulties they experienced).

One important topic of discussion during the debriefing interviews was with the problems all subjects faced while performing the tasks. Language and framework wise, the main sources of problems were: a lack of Struts experience (row *g*), a lack of GUI development experience in JSP/Struts (row *h*), and being out of practice with the Java programming language (row *i*). The UML subjects reported having more problems than the no-UML subjects in this area. Also, the UML subjects reported having more problems due to poor naming of variables or classes (row *l*). Lastly, problems in understanding specific areas of the system most often point to the *Best Codes Manager* (a complex part of the system discussed in Section 5.2.7.4). Half of the no-UML subjects reported problems understanding this part of the system compared to only two subjects in the UML group.

In terms of the subjects' opinion of the BESTweb system's quality (rows *a* to *c*), most in the no-UML group thought the system was better than average, as opposed to just two in the UML group. The one (UML) developer that considered the system below average was used to developing safety-critical systems. In terms of the BESTweb system's documentation (rows *d* to *f*), again, more no-UML subjects thought that the system documentation was above-average than their UML counterparts. The one (no-UML) developer that felt the documentation was below-average thought so due to the lack of any domain models in the documentation. Further, UML did not seem to have an effect on the effort spent on the architecture document (row *k*).

Table 5.11: Summary of qualitative results applicable to all subjects

Row	Code	no-UML	UML
(a)	In my opinion, the quality of the BESTweb system was below average compared to what exists in industry.	0/10	1/10
(b)	In my opinion, the quality of the BESTweb system was average compared to what exists in industry.	2/10	7/10
(c)	In my opinion, the quality of the BESTweb system was above average compared to what exists in industry.	8/10	2/10
(d)	In my opinion, the documentation that came with the BESTweb system was below average compared to what exists in industry.	1/10	0/10
(e)	In my opinion, the documentation that came with the BESTweb system was average compared to what exists in industry.	4/10	8/10
(f)	In my opinion, the documentation that came with the BESTweb system was above average compared to what exists in industry.	5/10	2/10
(g)	Reported that lack of Struts experience caused considerable problems.	5/10	8/10
(h)	Reported that lack of GUI development experience in JSP/Struts caused considerable problems.	5/10	5/10
(i)	Reported feeling rusty with the Java language.	0/10	3/10
(j)	Reported a lot of trouble understanding the *Best Codes Manager* (BCM) –related portion of the system.	5/10	2/10
(l)	Reported having trouble understanding parts of the system due to poor naming of variables or classes.	1/10	3/10
(k)	Reported only skimming the BESTweb architecture document (instead of thoroughly reading it).	6/10	6/10

Information offered by the no-UML subjects is presented in Table 5.12. Half of the subjects reported missing the presence of at least some models (e.g., the domain model). Two subjects reported drawing their own UML diagrams to aid with comprehension of the system. Half of the subjects also reported having problems grasping an overview of the functionality of interest in the system (two of these five did not report missing models). Two of the subjects stated that they felt that they would have gained no benefits from the presence of UML.

Table 5.12: Summary of qualitative results applicable solely to the no-UML subjects

Row	Code	Number of Developers
(a)	Reported missing models	5/10
(b)	Reported drawing own UML diagrams	2/10
(c)	Reported having problems gaining an overview of functionality	5/10
(d)	Specified that no benefit would be gained from the presence of UML	2/10

The qualitative results for the UML group, presented in Table 5.13, suggest that the extent to which UML documentation was used, and its impact, varied among the UML subjects. The experiment required that all subjects update the diagrams before they moved on to the next task. However, the extent to which they used the UML models to identify change locations prior to performing code modifications varied greatly among subjects.

First, in terms of types of UML diagrams used, rows *a* to *e* show that all subjects used the sequence diagrams, all but two used the use case diagram, half used the class diagrams and the page flow diagrams (these diagrams modeled the Struts/JSP elements' interaction with the rest of the application), while only one subject used the statechart diagram. Even though most subjects used a subset of the available diagram types, all said that they found UML generally useful (row *f*). Seven of the ten also said that they found that UML aided in getting an overview of the system (row *g*), even though two of these seven also stated they used the UML to a lesser extent (see descriptions for rows *h* and *i*).

Taking a closer look at the extent in which UML was used reveals that four subjects consciously limited their use of UML (row *h*). One of these four thought that taking advantage of the UML was optional in the case where he thought it would make him more efficient (thus, he chose not to use it "too much" to avoid wasting the client's time). Three of these four reported that the low use was due to anxiety that it would take more time (to complete the tasks), and, to a lesser extent, habit (row *i*). Further, two of these three (in row *i*) struggled with implementing the tasks, thus, the added burden of learning to use UML in such a manner was too much for them.

Rows *j* to *l* deal with the manner in which the UML was used and reveal that most subjects used the UML diagrams to navigate around the system, more than half used it to find the exact places in the code that needed to be modified, and only two did not look at the UML once they started development.

Table 5.13: Summary of qualitative results applicable to the UML subjects

Row	Code	Number of Developers
(a)	Took advantage of the Use Case Diagram	8/10
(b)	Took advantage of Sequence Diagrams	10/10
(c)	Took advantage of Class Diagrams	5/10
(d)	Took advantage of Page Flow Diagrams	5/10
(e)	Took advantage of the Statechart Diagram	1/10
(f)	Found the UML diagrams useful	10/10
(g)	Reported that UML was helpful in gaining an overview	7/10
(h)	Did not use UML to the maximum extent, reason: not used to working with UML	4/10
(i)	Did not use UML to the maximum extent, reason: did not want to risk wasting the client's time – a subset of (*h*)	3/10
(j)	Used the UML for navigation (browsing the system)	8/10
(k)	Used the UML diagrams to find exact code change places	6/10
(l)	Did not look at the UML diagrams during development	2/10
(m)	At some point was unable to gain an understanding of a piece of the system using UML	7/10
(n)	Reported problems comprehending the large Sequence Diagram (quantified in Table 5.4)	4/10
(o)	Reported problems comprehending the large Class Diagram (quantified in Table 5.5)	3/10
(p)	Did not trust the UML diagrams to be accurate	1/10
(q)	Reported needing Javadoc comments when looking at the diagrams	3/10
(r)	Found the UML tool to be problematic	9/10
(s)	Felt he completed the tasks faster thanks to the UML	2/10
(t)	Felt he completed the tasks slower due to the UML	2/10
(u)	Extensive experience of working with UML	1/10
(v)	Found the UML tool training adequate	10/10

Problems and difficulties experienced by the subjects while using the UML are covered in rows *m* to *r*. One of the most commonly reported frustrations was (row *m*) the inability to extract what they were looking for from the UML diagrams (at some point in time); three subjects explicitly expressed frustration that they had to go into the code (and leave the diagram) to look at comments (row *q*). This stemmed from three main issues: lack of

experience with understanding non-trivial UML diagrams (it is one thing to understand individual constructs like a message on a sequence diagram, and another to take these individual pieces of information and combine them into a complete mental model), lack of direct access to code-comments from the diagrams (class and association descriptions in the comments may be crucial to understanding a cluster of classes on a class diagram), and a lack of knowledge how a UML representation translates to code (e.g. composite aggregation). The last point was observed with respect to the statechart diagram; even though the subject understood the statechart diagram he could not see how this was implemented in the code due to a lack of familiarity with the state pattern [14] (note that the UML tool did not support the linking of the statechart diagram to code).

It is important to point out that the four developers that made lesser use of the UML (row *h*) are a subset of the seven (row *m*) aforementioned problems. The other commonly reported frustration dealt with the UML tool (row *r*); the main complaints were that it was very painful to update the diagrams and the presence of faults (bugs). Another commonly reported issue dealt with the largest sequence (row *n*) and class (row *o*) diagrams, which was reported by six subjects. Finally, it is interesting to point out that trusting the accuracy of the UML diagrams was not a problem (row *p*).

In terms of the subject's perception concerning time saving, including the time that it took to update the sequence diagram, two developers said that they felt that they finished faster thanks to UML (row *s*) and two said that they finished slower because of UML (row *t*). One of the two that said that UML slowed him down was the developer that got discouraged when UML did not help him as much as he thought it would on the first task. One developer said that he was "at least as efficient and [UML documentation would] make people coming later on the project more efficient. Could cut the time [it takes to catch up] in half."

Lastly, while only one of the subjects (row *u*) in the UML group had extensive experience in working with UML (worked in a similar manner to what was asked of the subjects in this experiment), the remaining subjects had training that can be considered representative of what most practitioners who qualify themselves as experienced in UML-based developers have. Thus, nine of the subjects were learning to use UML in such a comprehensive manner, and none of the ten had prior experience with the adopted UML tool. During the interviews all subjects agreed that the one day training that they received at the start of the experiment provided all the necessary information needed to use the tool,

but they all also said that that would have liked to have more training (to various extents). Thus, the training they received was adequate (row *v*).

5.3.4 Discussion of Qualitative Results

The qualitative results reveal the differences in experiences that the subjects had with the BESTweb system. Furthermore, subjects from the UML group provided insight into how they used UML and their opinions of working in such an environment. This section will look at these two issues using the presentation format from the previous section; starting with a discussion comparing the two groups, followed by a discussion on the no-UML group, concluding with a discussion pertinent to the UML group.

The first large difference between the two groups is the UML group's seeming disadvantage (to the no-UML) in terms of Struts experience and comfort with the Java language (Table 5.11, rows *g* and *i*). More subjects in the UML group claimed to have serious problems with Struts than did the no-UML subjects (8 vs. 5). This is significant as the BESTweb system is based on the Struts-framework, and this is a framework that has a productivity threshold. Thus, this could have negatively affected the overall performance of the UML group. Yet, we recruited people with the required background, so this was surprising. Taking a look at Task 3 provides some insight into this result since this was the first Struts-heavy task. The quantitative results do not point to the UML group having more problems. Thus, UML may have helped them deal with lower Struts experience. Further, the GUI shows that an equal number of subjects in each group are expressing problems with GUI development in Struts/JSP. This is further proof that the groups were probably well-matched, despite the self-analysis of their Struts expertise. Furthermore, three subjects in the UML group reported that they were rusty in Java. This is a serious issue as it means that the subjects' efforts were not solely focused on solving the tasks. All these observations suggest that our estimate of the impact of UML is probably a conservative, lower bound.

Next, in terms of the subjects' perception of the system, the no-UML subjects had a higher opinion of the system in terms of the system's quality and documentation (Table 5.11, rows *a* to *f*). The most plausible explanation for this may be that the UML documentation allows you to see more problems and therefore the UML subjects were more critical. Also, more UML developers complained of poor naming of variables and classes in the system than no-UML developers, three versus one (Table 5.11, row *l*). The UML developers said

that when they were looking at the sequence diagram, a poorly named variable or class made the diagram much harder to understand, forcing them to go to the code. This may have been less of an issue with the no-UML developers as they were already in the code and could immediately look at associated comments to get the explanation behind the name (the system was completely documented with Javadoc comments).

With respect to system comprehension, one specific part of the system, the Best Codes Manager (BCM) caused particular problems (Table 5.11, row *j*). The no-UML subjects reported having more problems understanding this portion of the system. This may indicate that the UML documentation aided in understanding this complex part of the system. Additionally, the BCM is one of the parts of the system that made Task 4 complex (as discussed in Section 5.2.7.4). The fact that the UML group took longer to complete Task 4, even without including the time they used on updating the UML, may point to the fact that they took the time to understand that portion of the system, resulting in fewer erroneous submissions.

The key discussion point unique to the no-UML subjects was the lack of models in the documentation. This discussion varied greatly as each subject's opinion was largely influenced by their previous experience using models (if any), and was completely out of our control. Keeping these points in mind, the qualitative results reveal that half of the subjects reported missing some kind of model representation of the system (e.g., the domain model) (Table 5.12, row *a*), two of these ended up drawing their own UML diagrams (both drew class diagrams and one also drew sequence diagrams) for the more complex parts of the system (Table 5.12, row *b*). Also, half of the subjects reported having problems gaining an overview of functionality (Table 5.12, row *c*), three of which also reported missing models (Table 5.12, row *a*).

The UML subjects all found the UML to be generally useful (Table 5.13, row *f*), even though they used the UML to various extents (at the very least, they had to update it). Curiously, even though they all found it useful, only one subject used the UML to its fullest extent: did not try to minimize the amount of time spent on UML, made use of all the diagram types and used the UML artifacts for navigating and locating code-change places. This varying use of UML amongst the subjects will now be examined, first in general terms and then in terms of specific types of diagrams.

Four of the subjects reported that they did not use UML to the maximum extent out of habit (Table 5.13, row *h*). Furthermore, three of these four subjects also said that they did not take as much advantage of UML as they could have for fear that it would take longer to solve the task in that manner (Table 5.13, row *i*). These subjects would use UML to get an overview of the system, but would then try to rely on the code to hasten the development time by, for example, checking a detail in the code rather than going back to the sequence diagram. This was unfortunate because if the developer had an incorrect solution to the problem, either because they did not understand the specifications or because they formulated an incorrect (mental) solution, they did not give UML a chance to help them arrive at the correct solution. One of the subjects did not like to rely on the UML, as he felt more confident with code. He only used parts of the UML that he found most useful, as opposed to trying to take advantage of all the UML documentation by reading and understanding it. Yet another subject spent a lot of time trying to use the UML on the first task. When he felt the UML did not help him solve the task this discouraged him from using it to the same extent on the remaining four tasks. Thus, we can conclude that the primary reason for lack of use of the UML in general is habit (or, lack of experience with UML) and fear of being inefficient. This is anecdotally confirmed by the fact that the only subject that took advantage of the UML to the full extent was the subject with extensive experience with UML (Table 5.13, row *u*).

In terms of the use of specific diagram types (Table 5.13, rows *a* to *e*), half of the developers took advantage of only two types of diagram. Most subjects used the UML artifacts for navigation purposes and in order to obtain an overview of the system (Table 5.13, rows *j* and *g*), this is confirmed by the fact that the use case and sequence diagrams were the most-used (Table 5.13, rows *a* and *b*). Unfortunately, two of the developers did not use the use case diagram (Table 5.13, row *a*). This is unfortunate as all other developers that did use the use case diagram reported that it was very beneficial. This is natural as it is the starting point from where the developer finds the use cases that need to be modified. These use cases are linked to the sequence diagrams that show which classes, objects, and methods are involved in the execution of the functionality specified by the use case. The class and the Struts-specific page flow diagrams (Table 5.13, rows *c* and *d*) were used by half of the developers, and the statechart diagram (Table 5.13, row *e*) was only used by one of the developers (the one that was highly experienced at using UML). When asked why they did not take advantage of these UML diagrams, they said that they either

did not feel the need or that since the other diagrams were much less integrated into the tool, it seemed troublesome to use them (not worth the effort). The fact that only half of the subjects used class diagrams was particularly surprising. While the class diagram is quite well integrated into the tool, the problem was the presence of too much irrelevant information at the same time (irrelevant for the specific task). Unlike in the case of sequence diagrams which inherently only display the objects and interaction pertaining to a specific use case, the class diagrams display a "package view". In this view the class diagrams display all the classes/associations in the package. This often leads to unnecessarily complex diagrams since system learning occurs in an iterative manner. When a developer needs to modify the functionality of a use case, they first need to focus only on the classes pertaining to that use case. This is very difficult when the classes are buried in a package view class diagram as they need to filter out the unnecessary parts of the diagram cognitively. A way to address this problem is by having a use case view of a class diagram. In this view only the classes/associations that are used by the accompanying sequence diagram would be visible. Further, class diagram views should also enable the creation of a special cluster of classes, for example, belonging to a design pattern.

The developers listed the following advantages when modeling:

- Traceability: The ability to quickly identify relevant parts of the system that need to be understood in order to implement a change (or determine the parts of the system they needed to understand in order to solve the task). With UML this is accomplished by first identifying the relevant use case (on the use case diagram) and then looking at the according sequence diagram.
- Visualization and abstraction through modeling can convey information that is hard to retrieve from the code:

  (1) Unnecessarily complex solutions are easily visible, e.g., if a sequence diagram is very large (cannot be easily viewed on the computer screen) then it may help if the solution was simplified by decomposing it into subsystems. This, in turn, creates a more modularized and easier to understand solution.

  (2) Composition is clearly seen, this helps to prevent the deterioration of the system's architecture (references will not be passed to objects that shouldn't have them).

  (3) All states and transitions are explicitly specified helping the developer understand "the big picture" faster.

(4) With sequence diagrams, comments can be made on the dynamic view of the system (as opposed to the comments found in the code that only relate to the system from a static point of view).

Conversely, the developers revealed the following frustrations that they experienced:

- It was very painful and problematic to create and update sequence diagrams. This is primarily a problem with the tool, and not UML itself (Table 5.13, row *r*).
- The UML diagrams were useful for understanding parts of the system that they were unfamiliar with, but, later, after gaining familiarity with the system, the developers thought that the UML was less useful overall, due to the overhead of creating/keeping the UML diagrams up-to-date. Though the diagrams were still useful at determining if the solution was good. Again, a lot of this overhead stemmed from the developers that had to struggle with the tool.
- When a developer tried to understand the system from the UML diagrams, yet failed to do so, they reverted to the code. They then felt that they wasted time on the UML diagrams.
- The large sequence diagram and the large class diagram were overwhelming (Table 5.13, rows *n* and *o*). It is interesting to note that the developer that was very experienced with UML reported having trouble with the large class diagram, but not with the large sequence diagram.

So, overall, we can see that the main problems faced by developers are either related to deficiencies in the tool or the need for further training and experience in using UML. This confirms further that the potential benefits of UML, in the mid and long term, are probably larger than what was observed in this experiment.

Based on the discussions with the subjects we also believe that the following items will help the adoption of UML in practice:

- A refined UML tool with which the developers do not need to struggle with (Table 5.13, row *r*). The tool should enable developers to use a subset of its functionality allowing for a gradual adaptation, see Section 5.6.
- A book on UML that, instead of describing the notation, draws on the best practices from experienced users of UML. This book would be analogous to the design Patterns book for Object-Oriented Programming [14] and the effective-series book for Java and

C++ [48, 100]. The book would deal with topics such as the following (raised during the training of the UML subjects):

- Heuristics for sequence diagrams, like the maximum number of elements that can appear on the diagram [101].
- When it is more appropriate to use the collaboration diagram instead of the sequence diagrams.

5.3.5 Summary of Results

In terms of time, the UML subjects used more time if the UML documentation were to be updated (though, slightly less if it were not). With the total time (*T*) the subjects spent on the five tasks, we see that the UML group completed the tasks slightly faster (1.4%) than the no-UML group (Table 5.7). This difference is not practically or statistically significant. When we take the time it takes to update the UML documentation into account, we see that the UML group spent 14.5% more time on the five tasks, though this difference is not statistically significant either and may therefore be due to chance. On average, the UML subjects spent 14.8% of the total time reading the UML documentation and 13.2% updating the documentation.

UML was always beneficial in terms of functional correctness (introducing fewer faults into the software). The subjects in the UML group had on average a practically and statistically significant 54% increase in the functional correctness of changes ($p=0.03$). UML also helped produce code of better quality when the developers were not yet familiar with the system. A significant difference was found for Task 1, the UML group's design quality score was 56.2% higher ($p=0.0025$). Though, across all the tasks, there was an insignificant 7% improvement in design quality ($p=0.22$).

All the qualitative evidence suggests that the observed impact of UML on change quality and productivity is probably very conservative in this experiment. The UML subjects were at a disadvantage when it came to Struts experience and familiarity with Java. We also observed that half of the subjects only used two diagram types, with the use case and sequence diagrams being by far the most used. Four of the subjects did not use the UML to the extent that they could have on account of concern that UML would make them less efficient, and habit (not being used to using UML). The subjects also experienced severe problems when dealing with the tool and in understanding the large sequence and class diagrams. However, the qualitative evidence also explains the observed benefits of UML.

The no-UML had more problems understanding a complex part of the system. All subjects found the UML to be generally useful: the largest benefits were traceability of use cases to code and the ability to quickly get an overview of the system.

The results of this experiment, both qualitative and quantitative can also be used to guide industrial adoption with respect to, at the very least, applications with similar properties (e.g., web applications). In the case of developers that are not very experienced using UML and that are to perform maintenance tasks on a system they are not familiar with, the use case diagram along with the sequence diagrams seem to be the most cost-efficient parts of UML. This appears to be the case for two reasons; first developers inexperienced with UML are overwhelmed by too many diagram types and will only use those that are easy to use. Next, these two diagrams help them quickly identify the relevant code for specific functionality needed to perform the maintenance tasks. Given these advantages, these two types of diagrams can also be considered a cost-efficient starting point for introducing UML into the organization.

5.4 Threats to Validity

The reported experiment is very realistic in general and in particular when compared to previously reported experiments on UML. In fact, the main strength of this experiment lies in its external validity: professionals worked on a real system, using real tools, implementing real tasks. Furthermore, the fact that the developers worked until the tasks were implemented correctly ensures that this experiment does not suffer from the construct validity problems with respect to correctness: how do you include unfinished or incorrect solutions in your analysis?

The hypotheses were formulated in such a way that the results obtained could be generalized to a target population of professional Java consultants performing real programming tasks with professional development tools in a realistic work setting. However, this is an ambitious goal; one that is difficult to achieve. For example, there is a trade-off between ensuring realism (to reduce threats to external validity) and ensuring control (to reduce threats to internal validity). This section discusses what we consider to be the most important threats to the validity of this experiment and offers suggestions for improvements in future experiments.

5.4.1 Statistical Conclusion Validity

Validity of statistical conclusions concerns (1) whether the presumed cause and effect covary and (2) how strongly they covary. For the first of these inferences, one may incorrectly conclude that cause and effect covary when, in fact, they do not (a Type I error) or incorrectly conclude that they do not covary when, in fact, they do (a Type II error). For the second inference, one may overestimate or underestimate the magnitude of covariation, as well as the degree of confidence that the estimate warrants [102].

Recall that the individual level of significance for the hypotheses tests were set to $\alpha = 0.05$. No significant differences were found with respect to the dependent variable time. While the effect size between time it took to implement the tasks without account for the time spent on updating the UML documentation (T) was negligible (1.4%), the effect size for time spent on implementing the tasks in total (T') was -14.5%. Furthermore, the effect size measure d is well below 0.5 for both T and T', and as explained in Section 5.2.8.1, this indicates a *small* treatment effect. Considering the fact that the number of subjects we used in the experiment was determined by budget constraints, it is illuminating to note that for us to find a statistically significant difference with 80% power, the UML group would have to have a mean of 45% larger or smaller than the no-UML group (using the UML group's variance). Thus, given the effect sizes and our sample size, we were unlikely to find a statistically significant effect.

5.4.2 Internal Validity

The internal validity of an experiment is "the validity of inferences about whether observed covariation between A (the presumed treatment) and B (the presumed outcome) reflects a causal relationship from A to B as those variables were manipulated or measured" [102]. If changes in B have causes other than the manipulation of A, there is a threat to the internal validity.

The main threat to internal validity in this experiment could have been the lack of random assignment to the two treatment groups: no-UML and UML. This was not possible due to practical reasons (see Section 5.2.4), thus making this a quasi-experiment. Fortunately the groups were in every practical aspect equivalent, as discussed in Section 5.2.4. Furthermore, an analysis of covariance was performed to adjust for the effects of grade, degree, and experience in terms of years and LOC written, all producing no different results (although grade and degree explained the variance best). Thus, we do not consider

this to be a major threat. Note that, during the debriefing interview, three subjects in the UML group reported that they felt rusty in Java (see Section 5.3.3). Also, three more subjects in the UML group felt that lack of Struts knowledge caused serious problems. So, if there is any imbalance between the UML and no-UML groups, it is to the detriment of the former.

5.4.3 Construct Validity

Construct validity concerns the degree to which inferences are warranted, from (1) the observed persons, settings, and cause and effect operations included in a study to (2) the constructs that these instances might represent. The question, therefore, is whether the sampling particulars of a study can be defended as measures of general constructs [102].

In the case of this experiment we examine three such constructs. First, to investigate the effects of UML, the subject must have actually used the UML; this is discussed in Section 5.4.3.1. Next, one of our dependent variables deals with software quality, a concept inherently without a precise definition [94]. In Section 5.4.3.2 we discuss the issues that are present in our definition of design quality, a small aspect of software quality. Last, no experimental setting can show the true cost of fixing a fault. The shortcomings of measuring fault-cost via the time-effort it takes the developer to correct the fault is addressed in Section 5.4.3.3.

5.4.3.1 Usage of UML

UML has many facets to it [24]: the choice of diagrams that are used, the level of detail of these diagrams, and the type of tool that is used (if any). In this experiment we use five types of diagrams (use case, sequence, class, statechart, and page flow) at the level of detail used in [12]. Also, the subjects were given a state-of-the-art UML development environment along with the printed UML documentation.

To ensure at least a minimum usage of the UML the following steps were taken: the UML subjects were given training in the UML tool, the subjects were encouraged to take advantage of the UML (and not simply to ignore it assuming that this way they would save time), and the UML had to be updated before the solution was accepted. Even with all of these precautions, half of the developers took advantage of only two types of diagrams (see the Appendix). This is not too surprising given that nine out of the ten UML subjects were new to such an extensive use of UML and there is a learning curve before a complete use of the artifacts can be made. In fact, only the subject that was highly-experienced with

using UML (in such a manner) took advantage of all the diagram types. Furthermore, the tool can be improved in many ways for both ease of understanding the existing design and ease of updating the artifacts (see Section 5.6). Consequently, the results of this study might be a conservative measure of UML's effectiveness, since the developers probably did not reach their maximum level of efficiency during the experiment. To address this threat, future experiments should consider using developers that have passed the learning curve of using UML in such an advanced manner (note that the necessary subjects with such a background could not feasibly be found at the time of this experiment). In fact, such experiments always need to select a trade-off between assessing the maximum potential benefit of UML or a realistic impact based on current, common skills. We chose to focus on the latter in this experiment as we feel that UML experiments with highly-trained students (in UML) give an insight into the former [57].

5.4.3.2 Design Quality

We have chosen to focus on a limited aspect of design quality, a subtopic of the broad topic on software quality. As discussed in Section 5.2.6, arriving at a score for the solutions' design quality involved: (1) breaking down each task into subtasks, (2) specifying whether a potential solution followed the proper OO principles [12], and (3) categorizing each subject's solution according to the scheme. Although this is only a small aspect of software quality, it was chosen as it met our two criteria: (1) it could be used to compare the solutions across all the subjects and (2) it is repeatable. While the decision regarding the type of code that follows proper OO design principle is to some extent subjective, the process that we used to determine if a solution is acceptable or not is repeatable as each solution clearly maps to a defined category (ensuring reliability). Furthermore, the granularity at which this analysis was performed could be finer (e.g., we could also evaluate the variable names used), but then it would not be easy to compare the quality of the solutions across all the subjects. The main reason for this was the fact that the developers changed the system in a substantial manner and had a lot of flexibility in the manner in which they extended the design.

5.4.3.3 Cost of Fixing Faults

In this experiment, the cost associated with correcting a fault was the amount of time that it took the developer to fix the specified fault. These faults were precisely pointed out to the subjects by the experimenters; the time spent by the experimenters to find the faults was not included in T and T'. This is not realistic as, in real-world scenarios, when an end-user

finds a fault; the costs are much greater than just the correction effort. In [5, 6] experts mostly agreed that, for severe defects, “finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.” Some of the reasons for this increase in cost are:

- The time to find the defect increases.
- There is a cost to package and deliver the fix.
- Potential cost to the customer due to downtime or data corruption.
- The company may suffer in terms of damage to reputation and share price.

5.4.4 External Validity

The issue of external validity concerns whether a causal relationship holds 1) over variations in persons, settings, treatments, and outcomes that were in the experiment and 2) for persons, settings, treatments, and outcomes that were not in the experiment [102].

The major strength of this study is in fact its external validity: the subjects were experienced professional software developers from various consulting companies that worked on a real, non-trivial system where they implemented real and also non-trivial change tasks, using a real development environment (IDE), during an extended period of time (compared to other empirical studies on UML). Of course, the fact that some minimal degree of control had to be exercised and thus, the situation did not ideally recreate “a normal day at the office” for the developers, was a tradeoff that had to be made to strike a balance between external and internal validity.

The scope of this study is limited to situations in which the developers have no prior knowledge of the system to be changed, and it is possible that the results do not apply to situations in which the developers are also the original designers. Also, it may be the case that the results do not apply to systems in different domains.

5.5 Related Work

In what follows, we contrast the results from this experiment with the results from other experiments and studies that have investigated, in the context of program comprehension and maintenance, the costs and benefits of using UML and the impact of program documentation. This chapter is divided into two sections, where a general overview of the related work is given in Section 5.5.1, while an additional piece of related work is

discussed separately in Section 5.5.2 due to its close relationship to the work presented above. This latter section also discusses the principles of replication in software engineering experiments and shows how our work fits in this context.

Note that even though the systematic review (Chapter 4) can be considered a formal related work section for this study; this section is not redundant as it presents additional relevant papers that fell outside of the scope of the systematic review. There is one exception, triggered by the fact that this experiment can be considered a differentiated replication of previous work [57] (discussed in Section 5.5.2). This is due to additional detail beyond what is provided in Chapter 4 being necessary to fully understand how the studies differ.

5.5.1 Overview

Another experiment was conducted to assess the qualitative efficacy of UML diagrams in aiding program understanding [103]. Fifteen subjects, whose UML expertise varied (six beginners, eight intermediate, and one expert), had to analyze a series of UML diagrams (with access to code) and complete a detailed 60-minute questionnaire concerning a hypothetical software system. Results from the experiment suggest that UML's efficacy in supporting program understanding is limited by (1) unclear specifications of syntax and semantics in some of UML's more advanced features, (2) spatial layout problems, e.g., large diagrams are not easy to read, and (3) insufficient support for representing the domain knowledge required in understanding a program. This experiment only concurs with point (2). Note that in this experiment a Struts UML profile was used to adequately model the domain knowledge.

Furthermore, a controlled experiment investigated how access to textual system documentation (the requirements specification, design document, test report, and user manual) helped when performing maintenance tasks [104]. The results indicated that having documentation available during system maintenance reduces the time needed to understand how to perform maintenance tasks by approximately 20 percent. The subjects who had documentation available also showed a better understanding and a more detailed solution to how to incorporate the change, when compared to those who had only source code available. The results also suggested that there is an interaction between the maintainer's skill (as indicated by a pretest score) and the potential benefits of the system documentation: the most skilled maintainers benefited the most from the documentation. Although this work is not directly relevant, it is still relevant as UML can be considered as

a form of documentation. Our experiment does not support the claim that UML decreases the time it takes to perform the tasks, nor that the most skilled maintainers benefited the most from the UML documentation. However, our experiment does confirm that the presence of additional documentation in UML form does give the developers a better understanding of the system (via better correctness results).

5.5.2 Replication

In software engineering, like in other sciences, no single study can fully answer a large fundamental research question. Huxley [105] notes that "…in science, as in common life, our confidence in a law is in exact proportion to the absence of variation in the result of our experimental verifications." This problem is addressed via experiment replications, that is, the repetition of an experiment where some variables may, or may not, vary. Replications are necessary until such a time when additional verifications carry no further power of confirmation. The question of the validity of replications is addressed in detail in [106]. The authors persuasively argue that "…given the human component and the rich variety of software and hardware technologies, it surely is beholden on the community to perform many, many such verifications". Furthermore, the authors emphasize that "only under exceptional circumstances should one-shot studies involving subjects be relied upon." This point of view is shared by Curtis [107]: "…results are far more impressive when they emerge from a program of research rather than from one-shot studies."

Subjecting theory to experimental test is a crucial scientific activity, but researchers must be sure of their results before relying on them to take action. Popper [108] noted that "We do not take even our own observations quite seriously, or accept them as scientific observation, until we have repeated and tested them." Unfortunately, although it is agreed by the scientific community that replication is a crucial aspect of the scientific method, it is not widely practiced in software engineering. A systematic review of controlled experiments in software engineering showed that only 18% of the experiments were replicated [109]. This problem is not unique to software engineering as noted by Collins [110] "For the vast majority of science, replicability is an axiom rather than a matter of practice." and Broad and Wade [111] "How much erroneous…science might be turned up if replication were regularly practiced, if self-policing were a more than imaginary mechanism?"

A replication can vary along three dimensions [106]: experimental method, tasks, and subjects. A basic finding replicated over several different methods carries greater weight, as stated by Brewer and Hunter [112]: “The employment of multiple research methods adds to the strength of the evidence.”

Second, experimenters must decide whether a similar or alternative task should be used. Again, a basic finding replicated over several different tasks carries greater weight. Curtis [107] stated, “When a basic finding…can be replicated over several different tasks…it becomes more convincing.”

Third, the subjects must be considered. Not surprisingly, a basic finding replicated over several different categories of subjects also carries greater weight. This is especially true in the context of software engineering where skills and experience have such an extensive influence on the cost-effectiveness of technologies [17, 113].

Table 5.14: Study Comparison – Experimental method

	Oslo/Ottawa	This Experiment
High-level Research Question	What is the impact of UML documentation on software maintenance?	
Experiment Design	Between- / Within-subject	Between-subject
Randomization Method	Blocking	No-blocking
Setting	Laboratory	Office
Population	Trained Students	Senior Professionals
Allocated Time to Complete All the Tasks	8 hours / 15 hours	Unbounded
Average Time Taken to Implement All Tasks (min)	202 / 492	2113
# of Tasks	4 / 4	5
Recruitment	Call for participation & financial incentive. / Part of coursework.	Call for participation & financial incentive.
Minimal Skills	3rd/4th year software engineering students having been exposed to UML	Professional software engineers with experience in UML along with other technologies and environments
Number of Subjects	20 / 78	20
UML Tool	TAU / Visio	Borland Together
System Type	Artificial	Real-world (in actual use)
# of Classes	7 / 12	50
# of Use Cases	8 / 5	16
Lines of Java Code	338 / 293	2921
Mortality Rate	9% / 14%	0%

This experiment can be a considered a *differentiated* [109] replication of two previous experiments [57], henceforth referred to as the Oslo/Ottawa experiment, since the same phenomena is studied but all three replication-dimensions are changed (maximizing the weight of the replication): experimental method, task, and subject. The experimental method itself is composed of various aspects related to the ways in which the studied constructs are measured, how the impact of the treatment (i.e., UML) is analyzed, and how the randomization of subjects is performed. These differences will now be presented in a rigorous and structured manner.

Table 5.14 outlines the main differences in terms of the experimental method whereas Table 5.15 describes the measurement of dependent variables separately, for the sake of improving the tables' legibility.

Table 5.15: Study Comparison - Dependent Variables

	Oslo/Ottawa	This Experiment
Time Spent on Development	Amount of time used by the subject until a solution is submitted, correct or not.	Amount of time used by the subject until a **correct** solution is submitted.
Time Spent on UML	Only time spent on updating UML documents is kept track of.	Time spent on both updating and reading UML documentation is kept track of.
Measurement of Correctness	Percentage of correct solutions. / Number of functional test cases passed.	Number of submissions before arrival at the correct solution.
Code Quality Appraisal Method	None. / The design quality of a task solution was assessed on the basis of counting the number of elements that were correctly and erroneously added, changed, and deleted, based on a pre-defined optimal solution.	A described in detail in Section 5.2.6, each possible solution for each subtask was rated as either *acceptable* or *unacceptable*, according to the pre-defined criteria following proper object-oriented design principles [12].

Though both studies are looking to evaluate and study the impact of using UML on software evolution, the manner in which they do this greatly differs as shown in Table 5.14 and Table 5.15. The Ottawa experiment used a larger number of subjects, and thus has greater statistical conclusion validity due to increased power. However, this experiment addresses the major weakness of the Oslo/Ottawa experiment, external validity, by the increase of realism. Thus, with respect to validity threats, the two studies are complementary.

Realism is increased in several ways. Professional software developers are used instead of students. The system that the developers work on is a real-world system instead of artificial, small systems. The settings are real-world (the developers had their own office) instead of a university laboratory setting. The tasks took much more time to implement. The UML tools used in the Oslo/Ottawa experiment, TAU [79] and Visio [80], were not as sophisticated and state-of-the-art as the one used in this experiment.

As opposed to the Oslo/Ottawa experiment, the developers in this experiment did not have time constraints. The drawback was that because of the larger cost incurred we could not recruit as many subjects as in the Ottawa experiment. But the absence of time constraints allowed us to ensure that the developers had to submit a functionally correct solution

before being allowed to proceed to the next task. Thus, time is measured differently in the two studies. Furthermore, we could also monitor more closely that the subjects followed experimental procedures throughout the experiment (the Ottawa experiment lost 11 subjects due to subjects not following instructions properly). In the Oslo/Ottawa experiment solutions were not checked for their correctness before being accepted. This is a crucial difference from the previous experiment as it makes the data analysis more reliable since we did not have to deal with partially correct solutions. In the Oslo experiment (which was in a way a pilot for the Ottawa experiment), correctness was measured by the percentage of correctly implemented tasks. In the Ottawa experiment, correctness was measured by the number of passed functional test cases. In this experiment, correctness was measured by the number of submissions (attempts) before arrival at the functionally correct solution.

Other differences include the fact that blocking was used in the Oslo/Ottawa experiment to ensure group equality. Although no blocking was performed in this experiment (due to practical reasons), a post-hoc analysis of the subjects data demonstrated that the groups were equivalent, a result we were expecting given our recruitment strategy.

In the Oslo experiment, the subjects implemented the four tasks in one day, for a duration of up to eight hours. The Oslo experiment had five change tasks that were completed during five course laboratories of three hours each, one task per laboratory, spaced a week apart.

Lastly, the design quality evaluation in the Ottawa experiment was at a finer degree than in this experiment due to the small size of the system and change tasks, by counting the number of operations, attributes that should be added, modified, or deleted based on each identified solution. No design quality evaluation was performed for the Oslo experiment.

The results of both studies are contrasted in Table 5.16. The results of two studies must be compared carefully due to the already discussed differences in research method and measurement of variables. For example, effort was measured in a different way. In Oslo/Ottawa effort equaled time “until *a* solution was submitted” while in this experiment equaled time until a *correct* solution was submitted.

Table 5.16: Study Comparison - Results

	Oslo/Ottawa	**This Experiment**	**Comment**
Time to Solve All Tasks *Excluding* Diagram Modifications	The no-UML group finished 25% slower. / The no-UML group finished 2.9% faster.	The no-UML group finished 1.4% slower.	Consistent: No significant differences.
Time *Including* Diagram Modifications	The no-UML group finished 27% faster. / The no-UML group finished 47.6% faster.	The no-UML group finished the tasks 14.5% faster than the UML group (not statistically significant).	Consistent: no-UML groups are faster.
Time Spent on Updating UML	35% / 30%	13.2%	Probably lower in this experiment due to the more experienced subjects using a more sophisticated tool. Also, subjects had more time to get used to the tool.
Correctness	Both experiments show that, for the most complex task, the subjects who used UML documentation performed significantly better than those who did not.	The UML group had 50% to 100% fewer faults in each and every task, and 54.7% fewer faults overall.	These results are consistent as all tasks in this experiment were complex, relatively speaking.
Design Quality	None. / UML solutions were better designed in terms of *correctly changed elements* (the average statistically significant difference was 0.50 on a five-point scale). With respect to *incorrectly changed elements,* a statistically significant difference was also found (average difference = 1.34).	Overall, the UML group had 7.3% more acceptable solutions.	Differences may be due to two factors: (1) the subjects in this experiment were experienced software developers and (2) the subject had to refine the solution until it was functionally correct before it was accepted. This may have resulted into a stronger convergence of the design quality of UML and no-UML groups.

First, in terms of effort, Oslo/Ottawa reports that "When considering only the time required to make code changes, using UML documentation does help to save effort overall." This is largely consistent with the results in this experiment: on average the UML subjects spent less time, but, the results in this experiment were not significant, perhaps due to lack of statistical power. Next, Oslo/Ottawa reports that "When including the time necessary to modify the diagrams, no savings in effort are visible." In fact, in both studies the no-UML groups finished faster.

In terms of time spent on updating the UML documentation, the Oslo/Ottawa experiment reports an overhead of 35% and 30%, respectively. This is higher than the overhead in this experiment (13.2%). We believe that this is due to this experiment having more experienced subjects, using a more sophisticated tool, for a longer duration (having had more time to get used to the tool). In terms of time spent on updating the UML documentation, the Oslo/Ottawa experiment did not collect quantitative data, but reports that "Most people [*thought* that they] spent less than 25% of the time in laboratory sessions understanding UML diagrams, over all tasks." In this experiment, on average, the subjects spent 14.8% of their time understanding the UML documentation (see Section 5.3.1.1).

In terms of correctness, in Oslo/Ottawa "…both experiments show that, for the most complex task, the subjects who used UML documentation performed significantly better than those who did not." These results are similar: we saw significant benefits of UML in terms of correctness.

Design quality was investigated in the Ottawa experiment where it was found that "… using UML helped to achieve changes with superior design quality, which would then be expected to facilitate future, subsequent changes." Across all the tasks, this is inconsistent with our results, though it is consistent with the results on Task 1. This may be due to two factors: (1) the subjects in this experiment were experienced software developers and (2) they had to refine the solution until it was functionally correct before it was accepted. This may have resulted into a stronger convergence of the design quality of UML and no-UML groups.

In summary, it is interesting to note that the results of the two studies show many similarities, despite the studies being very differentiated (differences in experimental method, tasks, and subjects). Thus, because we obtain similar results using different measurement, both with trained students and professionals, and systems of widely varying size, we can be confident that UML will bring practically significant benefits in a large number of conditions.

5.6 Suggested Improvements to the UML Tool

During the course of this experiment, from its design to the debriefing interviews with the subjects, ideas emerged with respect to possible improvements to the UML-tool we used (BTE) [85] in terms of usability, code generation, rule enforcement with respect to class and sequence diagrams, and means of gradual adoption of UML tools into widespread use.

Although these recommendations are targeted at BTE, most probably they also apply to other UML tools.

First, the subjects in the UML group expressed the need to return to the code to read comments. These comments can be made available directly on the diagram, saving the developer the need to go into the code, as that is highly distracting. One way of making these code comments available on the diagram is via tooltips. For example, on the class diagram, hovering over a class would bring-up a tooltip containing its corresponding Javadoc comments. Hovering over a method in a class or an association would bring-up its comment. On the sequence diagram, a tooltip with the code comment would appear when hovering over an object, a method, or a message. On use case diagrams, when hovering over a use case, a tooltip can show its description.

In the case of sequence diagrams, the following improvements to UML tools are highly recommended:

- When scrolling down on a sequence diagram, the objects should never disappear from view.
- The updating of existing diagrams can be facilitated by: (1) being able to add a message to an existing sequence diagram (in the appropriate place) directly from the code view, (2) the option to use dynamic analysis on the corresponding use case (with the appropriate filters being applied) and the (new) missing elements being automatically shown to the developer. The new additions can then be accepted or rejected from being displayed on the sequence diagram.
- It should be possible to generate a sequence diagram from dynamic analysis. Even though this is difficult to accomplish in a complete manner [32], an incomplete diagram that the developer could later refine would still be helpful.
- The developer should be able to ask the tool to display elements that are absent on the sequence diagram, and then selectively choose to add elements that should appear on that sequence diagram.
- Upon selecting an object or method on the sequence diagram, the corresponding class (and method) should be highlighted in a different color on the class diagram so that the developer saves time when looking for it.

In the case of class diagrams, as discussed in Section 5.3.4, support for views is necessary to help developers focus on the important parts of a class diagram (a view on a class diagram would only show the classes and associations of interest). Next, in addition to the ability of visually specifying composite relationships and immutable (frozen) classes, safeguards can be put in place to ensure that these rules are not violated. Also, generation of the clone and equals method can be largely automated when this information is explicitly specified (composition and immutability). This is important as these methods are very tricky to implement correct, as discussed in [48].

In terms of additional UI support for class diagrams, a specific use of the class diagram was deemed as potentially being very useful: the possibility for an IDE to show a subset of a class diagram where the only classes that would be displayed are: (a) the class of interest (e.g., being currently selected/modified) and (b) the class with which the class in (a) has immediate relations (that is, classes that exchange messages in sequence diagrams). This could also be used to gradually introduce UML into the development environment and let developers gradually adopt UML. Furthermore, this tool could also be used for visual dependency analysis. On a related note, a major inconvenience in the tool that was used in the experiment was the fact that the usage dependencies had to be specified manually, this is an unnecessary burden placed on the developers.

An investigation into a competing tool, IBM's Rational Software Architect [114], revealed that it implements only one of the suggestions presented here: when scrolling down on a sequence diagram, the objects never disappear from view.

5.7 Summary

An experiment was conducted to investigate the costs and benefits associated with UML during maintenance and evolution. This is the first such experiment performed on a real system, using professional developers as subjects and working with a state-of-the-art UML tool during an extended period of time. This chapter provides very clear insights in terms of the kinds of (minimum) benefits that can be expected from using UML and the factors limiting or boosting such benefits. In turn, such information can be used to decide about whether and how to introduce UML in a development organization. Although experiments are needed in other contexts as well (e.g., use of UML during initial development), we focused on software maintenance and evolution, by a non-original developer, as this consumes the majority of the resources in a typical software organization.

The quantitative results show that UML did not have a significant impact on the time that it took to perform the change tasks, both excluding and including the time it took to update the UML documentation. However, in terms of the functional correctness of the changes, UML had a practically and significantly positive impact, despite the fact that the UML subjects were not experts in UML and encountered many problems with the modeling tool. Lastly, in terms of design quality, a post-hoc analysis revealed a significant difference in the first task where the UML subjects were not yet familiar with the system and delivered solutions of higher quality. However, significant differences were not observed on the remaining four tasks. The qualitative results explained the probable root causes of the observed benefits: traceability from functionality to code and an abstract overview of the system structure and functionality. It also provided evidence that the observed benefits of using UML were probably conservative and that better tools and even more experience would likely yield a larger return on investment.

In terms of related work, the results largely support those of similar experiments, especially [57]. Because we obtain similar results using different measurement, both with trained students and professionals, and systems of widely varying size, we can be confident that UML will bring practically significant benefits under a large number of conditions.

6 Conclusion and Future Work

UML is subject to much controversy and debate when it comes to whether and how it should be employed during software development, where the opinions range from models ultimately succeeding code to models being ineffective. UML receives a lot attention for primarily two reasons; first, it would seem logical that since modeling helps so much in other engineering disciplines then it should also help in software engineering. Next, many see modeling in software engineering as part of the next *raising of the abstraction level,* helping deal with the growing complexity – the next paradigm shift.

Determining whether the use of UML can make a practically significant difference is therefore important. Unfortunately, comparing two software development paradigms is not as easy as, say, comparing two competing algorithms. Special methods must be used that take into account the many variables – human and technical – that make such comparisons so difficult.

While an investigation on this subject can look at various topics, this thesis focuses on the effects that UML has on developers' output during software maintenance and evolution. This was selected as it is the largest and costliest development phase and therefore has the opportunity for greatest return on investment.

This thesis contributes to the body of research by performing an extensive systematic review on the topic, aggregating and synthesizing published knowledge in the main journals and conferences on the topic. Furthermore, a large controlled experiment was conducted filling an important gap in the published work. Last, since the systematic review and the controlled experiment faced specific problems, existing methodologies for systematic reviews and controlled experiments had to be tailored to our context. Such methodological contributions should help performing future systematic reviews and realistic experiments in this problem area.

6.1 Summary of Results

The goal of this thesis was to answer the following six questions with respect to investigated topic using empirical evidence:

1. What are the costs, risks, and benefits of using UML?
2. What are the most effective ways of using UML?
3. What are the experiences of working with commercial UML tools?

4. Learning curve: how hard is it to learn UML in practice?
5. How is UML being used in the industry?
6. What are the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry?

These questions are answered based on the insights gained from both the systematic review and the controlled experiment.

What are the costs, risks, and benefits of using UML? This is treated as the most important one in the thesis and is therefore given the most attention. It was found that UML can be beneficial in terms of communication and documentation, correctness, testing, design, and sometimes effort. Costs were found to be training of staff, purchase and integration of tools, and sometimes additional effort due to diagram construction. The risks included misinterpretations of inconsistent and incorrect models.

Four papers in the systematic review discussed the effects of UML on correctness. While all specified positive gains, two concrete figures are given in terms of effect size that UML has in this area: "improved quality by 73%" and "typical results for quality improvements were a 1.2X–4X overall reduction in defects". This is confirmed in the controlled experiment; UML was always beneficial in terms of functional correctness (introducing fewer faults into the software). The subjects in the UML group had on average a practically and statistically significant 54% increase in the functional correctness of changes.

It seems that UML's effect on effort depends on the context, tools, code generation, and the size of the task. In two of the papers in the systematic review [55, 57] development with UML was less productive, but, in the study that was within an industrial context, it was revealed that thanks to time saved on other activities (fault correction), the time to market was the same [55]. It is worth noting that in [57] the tasks were small and the UML overhead was quite large for this system and these tasks. In [56] and [25] the productivity gains were quite large, though the results are confounded by the simultaneous introduction of OO technology in the former and model-based code generation in the latter. In the controlled experiment it was found that the UML subjects used more time if the UML documentation were to be updated (though, slightly less if it were not). Specifically, the UML group completed the tasks slightly faster (1.4%) than the no-UML group. This difference is not practically or statistically significant. When the time it takes to update the UML documentation is taken into account, the UML group spent 14.5% more time on the

five tasks, though this difference is not statistically significant either and may therefore be due to chance. It is probable that overall, effort will not be significantly increased by the adoption of UML and as more advanced tools are implemented (e.g., code generation) the gains will increase.

In terms of UML's effect on design quality, three observations were made on the topic in the systematic review: model-driven development lead to more focus on design, it was easier to have a larger team working on the same application, and superior design quality was achieved (maybe due to the fact that overly complex design is more easily visible on diagrams than in the code). In the controlled experiment it was found that UML also helped produce code of better quality when the developers were not yet familiar with the system. A significant difference was only found on the first task, the UML group's design quality score was 56.2% higher. Though, across all the tasks, there was an insignificant 7% improvement in design quality.

UML's effect on testing was not investigated in the controlled experiment, but the systematic review found that UML did have a positive effect. It was found that testing can be attributed to the diagrams being an easier and more structured starting point for test generation. Further, the models allow test generation and ensure completeness.

In terms of communication and documentation benefits, the three papers in the systematic review that discuss the issue all note the positive effects that UML had. This is not surprising as one of the goals of UML was to give developers a common modeling language with which they could communicate more effectively. Though communication was not investigated in the controlled experiment, UML artifacts were found to be a superior form of documentation thanks to support for traceability from functional requirements to code and the visualization of concepts that are difficult to extract from code, like composition relationships and the visibility of unnecessary complexity.

What are the most effective ways of using UML? In the systematic review, the papers on the topic looked at different aspects to this question: dynamic modeling, constraints, measurement and predication, reverse-engineering of code to diagrams, and use case authoring. The results can be summarized as follows:

- Clear and positive results were found for modeling constraints, specifically the employment of stereotypes, modeling conventions, and UML to code mappings.

- Deriving UML diagrams from existing code was shown to be more efficient and effective with a GUI-based approach versus a code-centric approach.
- Managing change during use case evolution was found to be a problem and solved using a *use case deltas* approach. A *use case delta* is defined as any change in a use case description that results from the addition, change or deletion of functionality described in a use case. A delta includes the critical information that needs to be specified. A delta specifies three pieces of information: the use case it relates to, a description of the change to the use case, and the source that triggered the change.
- Only the use case estimation method shows enough maturity to be used in practice [70].
- Introducing UML into legacy system during maintenance showed to be more difficult than on projects that used UML from inception.
- No clear results were found with respect to dynamic modeling and, with the exception of use case effort estimation. Possible reasons for this are addressed in the following section – Future Work.

In the controlled experiment it was found that the answer to this question is related to the amount of experience the developer had with UML: even though the developers in the experiment had access to many different types of diagrams, most of them focused only on use case diagrams and sequence diagrams. They found that the traceability afforded by the direct connection of the functional requirements in the use case diagram to the relevant objects and messages in the sequence diagrams was all they needed. Only the most experienced UML user took advantage of all the diagrams.

What are the experiences of working with commercial UML tools? Results from both the systematic review and the controlled experiment show that current UML tools are still lacking a lot of essential features. Basic functionality like model-consistency checking, model/code synchronization, and good model-editing support has not been reached. The tools are not user friendly and do not support the manner in which developers actually work with UML. For example, the developers in the controlled experiment found that synchronizing an existing sequence diagram with code changes to be very cumbersome – much more so than it needed to be due to lacking essential features. The tool assumed that changes would start from the sequence diagram, not the code – an unrealistic assumption.

Learning curve: how hard is it to learn UML in practice? UML's learning curve was in three of the papers covered by the systematic reviews, and the results were confirmed by

the controlled experiment: all consistently showed that UML is not trivial to learn. While it may be easy to be mislead into thinking that because UML is a visual notation it is therefore easy, in fact it takes experience at using UML before a developer can take full advantage of it. This is an important fact to acknowledge so that UML is not thrown at developers who have not been adequately prepared first with serious training. Organizations should also probably not expect large benefits from using UML on their very first project.

How is UML being used in the industry? The systematic review reveals that UML appears to be used in various technical roles and domains, on projects of various cost and size, by small and large teams. While the most common primary objectives for UML are to communicate requirements and to help guide development of code, more advanced goals are also reported like test and code generation.

An industry survey showed that most use UML sporadically, though 25% always use it. This may be related to the fact that only a third of the respondents specified that UML has management's full endorsement; this is a problem as the technology is too complex for large gains to be made by passive management practices. Not surprisingly, the most commonly used diagram types are use case, class, and sequence and the most commonly used tool is Rational Rose (though this information is probably dated). The results also showed that UML is methodology agnostic and is used in industry for advanced purposes like code and test generation.

What are the main issues that need to be addressed if UML is to be successfully implemented and widely adopted in industry? For UML to be successfully implemented and widely adopted in industry it needs to be supported by adequate processes and tools. The process must take into account that adopting UML is a non-trivial undertaking. Staff must have serious training, user friendly tools that support the manner in which developer actually work with UML, and commitment from management – an important ingredient to the successful adoption of any non-trivial technology. Tailored processes must be used for UML adoption in new system and legacy systems, where adoption poses additional challenges. Last, UML tools must be able to handle advanced tasks like handling a large number of models and use case evolution.

6.2 Future Work

The future work is discussed in four parts: holes in the existing research as identified by the systematic review, the shortcomings of existing UML tools, how the systematic review can be extended, and how the controlled experiment can be complemented.

The systematic review found that, with respect to the effectiveness of dynamic modeling, most of the papers compared sequence diagrams with collaboration diagrams in terms of effort and correctness. But the aggregated results failed to show general trends and this is probably due to the wrong kind of question being asked. The decision as to which diagram should be used, sequence or collaboration, depends on the properties that the diagram should highlight. This is an engineering decision where tradeoffs must be made and where professional experience comes into play. Thus, instead of asking "What is the best diagram – sequence, collaboration, or state?" it would be better to acknowledge the fact that these three diagrams should be used in different situations. Future experiments should ensure that they use subjects with proper training and a proper educational background, and focus on the following questions instead:

- What level of details should sequence/collaboration diagrams contain? Too much detail and the developers are overwhelmed and may find it easier to work with code (diagrams get abandoned). Further, diagram level of detail has direct implications on diagram-code synchronization.
- Anecdotal evidence suggests that developers find little use in diagrams that are too simple or too complex. This should be investigated to see with what kind of diagrams, with respect to complexity, most developers can work most effectively with.
- What is the most effective way of combining and using the different UML diagrams? This question should be asked in terms of both the presence of other diagrams and the user interface of UML tools. For example, how much more useful are interaction diagrams when class diagrams are also present? And, how should a UML tool's user interface look like so as to maximize usage efficiency?

Next, attempts have also been made at exploiting UML models for predictive purposes. Results from experiments with predictive models based on metrics for class diagrams and OCL show potential but more research is needed as the methods are still immature [69]. Subsequently, reported problems experienced during the adoption of UML show the need

for processes for both a full and a gradual adoption of UML into new and existing systems. Lastly, more papers from industry are needed to shed light on the type of problems practitioners actually face. These papers must not omit information pertaining to the UML tool use and the types of diagrams that were used, for what purposes, and the level of detail specified on the diagrams.

UML tools were found to be lacking in the following areas: user friendliness – not supporting the way that developers actually use UML, model-consistency checking, model/code synchronization, model-editing support, support for OCL, model retrieval support for reverse engineering, and tool interoperability. It is essential that these problems are solved by researchers and implemented by the tool vendors. Once these essential goals are reached, tool vendors can then focus on more ambitious goals like test generation.

Finally, the systematic review can be ameliorated by increasing the number of sources that it covers and, in the future, updated with relevant publications in the included sources. The controlled experiment should be replicated, preferably with more subjects that use more recent and sophisticated UML tools.

6.3 Concluding Remarks

When thinking about modeling a software system with UML, one can see the potential advantages like traceability from functional requirements in a use case, to specific object interactions in a sequence diagram, to code. Further, modeling allows for the developer to think, reason, and reflect about the system at a higher level of abstraction than the code. This thought and reflection can aid in the development of a better system. Another benefit of UML is that it gives developers a more powerful way of documenting OO code. Due to polymorphism and the decentralized manner of writing OO software, proponents of procedural programming argue that OO programs are more difficult to understand. While this may be true, it is interesting to notice that documenting the logic with a sequence diagram accounts for polymorphism – by showing objects, and the small methods, by continuously listing the logic in the same manner as would be seen in a procedural program.

UML then clearly has something to contribute; the question is whether the benefits of UML outweigh the costs and risks. Due to the number of variables in answering that question, empirical methods must be employed. Exactly this has been done in this thesis and the results suggest that indeed, the benefits do outweigh the costs and risks.

It is highly probably that gains will be even higher with tool support that not only efficiently supports the manner in which developers use UML, but also has advanced code and test generation features (as even without such tool support benefits are observed).

Lastly, even though experiments like the one in this thesis are very costly and labor intensive (this experiment took three years to prepare, run, analyze, and write-up), they are crucial and well worth the cost and effort in order to rationalize the manner in which new techniques are adopted in software engineering.

Appendix A Multivariate Analysis

Given the small number of subjects, we also fitted multivariate Analysis of Covariance (ANCOVA) models for the time and correctness data across all tasks and subjects. The average grade in computer science courses of each subject was included as a covariate to adjust for individual differences between the subjects and UML and Task (and their interaction) were the independent variables. The use of the covariate, combined with the fact that we had a total of 100 data points for each dependent variable (20 subjects and five tasks), resulted in increased statistical power compared to the less sophisticated univariate analyses. However, since the observations of individual tasks for a given subject are correlated, the ANCOVA assumptions of independent observations would be violated. We thus resorted to a statistical technique known as Generalized Estimating Equations (GEE) [98] to estimate the parameters (i.e., the effect of *Grade* – covariate to increase power, *UML* and *Task*) of the models. GEE is an extension of Generalized Linear Models, developed specifically to accommodate data that is correlated within clusters (here being the individuals).

The three complete model specifications are presented in Table B.1. Figure B.1 shows the corresponding SAS [115] code for two variations of the model: unreduced (a) and reduced or final (b). First, unreduced models were used, i.e. including interaction terms UML*Task, Task*Grade, UML*Grade, and UML*Grade*Task. In all cases, none of the interaction effects were significant, we thus reran the model without the interaction terms – see Table B.2.

When accounting for different task times while adjusting for within subject correlation using the GEE model, no significant differences were found between the UML/NO UML groups. Only the number of submissions, corresponding to correctness, is statistically significant – see Tables B.3-B.5. The results of the GEEs are thus entirely consistent with the univariate results.

Table B.1: Complete Model Specifications

Model	Response	Distrib.	Link	Model Term	Primary use of model term
1	Effort Excluding Updating UML	Gamma	Log	Grade	Covariate to adjust for individual skill differences
				UML	Models the effect of UML on effort
				Task	Models the effect of the task number on effort
2	Total Effort	Gamma	Log	Grade	Covariate to adjust for individual skill differences
				UML	Models the effect of UML on effort
				Task	Models the effect of the task number on effort
3	Number of Submissions	Normal	Log	Grade	Covariate to adjust for individual skill differences
				UML	Models the effect of UML on the number of submissions
				Task	Models the effect of the task number on the number of submissions

1a	`proc GENMOD data=A.subjecttask DESC;`
	`CLASS UML Task Subject;`
	`model Task_Dev_Time_ExUML = Avg_Grade|UML|Task / ALPHA=0.05 type3 LRCI LINK=LOG`
	`DIST=GAMMA;`
	`repeated subject=Subject / type=exch covb corrw;`
	`lsmeans UML Task/ diff cov;`

1b	`proc GENMOD data=A.subjecttask DESC;`
	`CLASS UML Task Subject;`
	`model Task_Dev_Time_ExUML = Avg_Grade UML Task / ALPHA=0.05 type3 LRCI LINK=LOG`
	`DIST=GAMMA;`
	`repeated subject=Subject / type=exch covb corrw;`
	`lsmeans UML Task/ diff cov;`

2a	`proc GENMOD data=A.subjecttask DESC;`
	`CLASS UML Task Subject;`
	`model Task_Dev_Time = Avg_Grade|UML|Task / ALPHA=0.05 type3 LRCI LINK=LOG`
	`DIST=GAMMA;`
	`repeated subject=Subject / type=exch covb corrw;`
	`lsmeans UML Task/ diff cov;`

2b	`proc GENMOD data=A.subjecttask DESC;`
	`CLASS UML Task Subject;`
	`model Task_Dev_Time = Avg_Grade UML Task / ALPHA=0.05 type3 LRCI LINK=LOG`
	`DIST=GAMMA;`
	`repeated subject=Subject / type=exch covb corrw;`
	`lsmeans UML Task/ diff cov;`

3a	`proc GENMOD data=A.subjecttask DESC;`
	`CLASS UML Task Subject;`
	`model Task_Num_Of_Submissions = Avg_Grade|UML|Task / ALPHA=0.05 type3 LRCI`
	`LINK=LOG DIST=NORMAL;`
	`repeated subject=Subject / type=exch covb corrw;`
	`lsmeans UML Task/ diff cov;`

3b	`proc GENMOD data=A.subjecttask DESC;`
	`CLASS UML Task Subject;`
	`model Task_Num_Of_Submissions = Avg_Grade UML Task / ALPHA=0.05 type3 LRCI`
	`LINK=LOG DIST=NORMAL;`
	`repeated subject=Subject / type=exch covb corrw;`
	`lsmeans UML Task/ diff cov;`

Figure B.1: SAS Code

Table B.2: Unreduced and Final Models

	Unreduced Model				Reduced (Final) Model	
Model	Model Term	DF	Chi-Square	Pr > ChiSq	Chi-Square	Pr > ChiSq
Effort Excluding UML	Grade	1	4	0.0454	3.46	0.063
	UML	1	1.27	0.2589	0	0.9661
	Task	4	8.29	0.0816	15.19	0.0043
	Grade*UML	1	1.6	0.2056		
	Grade*Task	4	5.9	0.2064		
	UML*Task	4	6.99	0.1366		
	Grade*UML*Task	4	6.26	0.1802		
Total Effort	Grade	1	4.19	0.0408	3.92	0.0477
	UML	1	2.22	0.1366	1	0.318
	Task	4	9.48	0.0501	15.48	0.0038
	Grade*UML	1	2.03	0.154		
	Grade*Task	4	7.04	0.1336		
	UML*Task	4	6.94	0.1392		
	Grade*UML*Task	4	6.8	0.1469		
Number of Submissions	Grade	1	4.73	0.0296	4.03	0.0447
	UML	1	1.4	0.2375	3.76	0.0524
	Task	4	3.11	0.54	9.15	0.0574
	Grade*UML	1	2.96	0.0853		
	Grade*Task	4	2.48	0.6481		
	UML*Task	4	3.47	0.4826		
	Grade*UML*Task	4	3	0.5582		

Table B.3: SAS Results for 1b – Effort Excluding Updating UML

Analysis Of GEE Parameter Estimates							
Empirical Standard Error Estimates							
Parameter		**Estimate**	**Standard Error**	**95% Confidence Limits**		**Z**	**Pr > |Z|**
Intercept		5.1707	0.5397	4.1129	6.2285	9.58	<.0001
Avg_Grade		0.395	0.2141	-0.0247	0.8147	1.84	0.0651
UML	No UML	0.0078	0.1817	-0.3483	0.3639	0.04	**0.9657**
UML	UML	0	0	0	0	.	.
Task	1	-0.7803	0.1233	-1.022	-0.5386	-6.33	<.0001
Task	2	-0.8984	0.1691	-1.2299	-0.5669	-5.31	<.0001
Task	3	0.2967	0.1311	0.0397	0.5537	2.26	0.0237
Task	4	-0.6154	0.1331	-0.8763	-0.3545	-4.62	<.0001
Task	5	0	0	0	0	.	.

Table B.4: SAS Results for 2b – Total Effort

Analysis Of GEE Parameter Estimates							
Empirical Standard Error Estimates							
Parameter		**Estimate**	**Standard Error**	**95% Confidence Limits**		**Z**	**Pr > |Z|**
Intercept		5.3706	0.5127	4.3656	6.3755	10.47	<.0001
Avg_Grade		0.3827	0.1985	-0.0063	0.7717	1.93	0.0538
UML	No UML	-0.1808	0.1784	-0.5304	0.1688	-1.01	**0.3109**
UML	UML	0	0	0	0	.	.
Task	1	-0.6823	0.1141	-0.906	-0.4586	-5.98	<.0001
Task	2	-0.983	0.155	-1.2867	-0.6793	-6.34	<.0001
Task	3	0.3338	0.1268	0.0852	0.5823	2.63	0.0085
Task	4	-0.6052	0.1257	-0.8517	-0.3588	-4.81	<.0001
Task	5	0	0	0	0	.	.

Table B.5: SAS Results for 3b – Number of Submissions

Analysis Of GEE Parameter Estimates							
Empirical Standard Error Estimates							
Parameter		**Estimate**	**Standard Error**	**95% Confidence Limits**		**Z**	**Pr > |Z|**
Intercept		-0.1927	0.3474	-0.8737	0.4882	-0.55	0.5791
Avg_Grade		0.4495	0.1271	0.2003	0.6986	3.54	0.0004
UML	No UML	0.31	0.1351	0.0451	0.5749	2.29	**0.0218**
UML	UML	0	0	0	0	.	.
Task	1	-0.1303	0.1407	-0.4062	0.1456	-0.93	0.3546
Task	2	-0.5249	0.2297	-0.9751	-0.0747	-2.29	0.0223
Task	3	-0.3243	0.2254	-0.7662	0.1176	-1.44	0.1503
Task	4	0.1313	0.1993	-0.2593	0.5218	0.66	0.5101
Task	5	0	0	0	0	.	.

Appendix B Detailed Qualitative Analysis Results

Table A.6.1: Qualitative analysis codes applicable to the no-UML subjects

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Total
Reported that lack of Struts experience caused considerable problems.	1		1	1	1	1					5
Reported that lack of GUI development experience in JSP/Struts caused considerable problems.		1	1	1	1		1				5
In my opinion, the quality of the BESTweb system was below average compared to what exists in industry.											0
In my opinion, the quality of the BESTweb system was average compared to what exists in industry.			1				1				2
In my opinion, the quality of the BESTweb system was above average compared to what exists in industry.	1	1		1	1	1		1	1	1	8
In my opinion, the documentation that came with the BESTweb system was below average compared to what exists in industry.						1					1
In my opinion, the documentation that came with the BESTweb system was average compared to what exists in industry.		1	1	1			1				4
In my opinion, the documentation that came with the BESTweb system was above average compared to what exists in industry.	1				1			1	1	1	5
Reported feeling rusty with the Java language.											0
Reported a lot of trouble understanding the *Best Codes Manager* (BCM) –related portion of the system.				1			1	1	1	1	5
Reported only skimming the BESTweb architecture document (instead of thoroughly reading it).	1		1	1			1	1		1	6

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Total
Reported having trouble understanding parts of the system due to poor naming of variables or classes.		1									1
Reported drawing own UML diagrams				1						1	2
Reported missing models			1	1	1	1				1	5
Specified that no benefit would be gained from the presence of UML	1	1									2
Reported having problems gaining an overview of functionality				1		1		1	1	1	5

Table A.6.2: Qualitative analysis codes applicable to the UML subjects

	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	Total
Reported that lack of Struts experience caused considerable problems.	1	1		1		1	1	1	1	1	8
Reported that lack of GUI development experience in JSP/Struts caused considerable problems.	1		1			1			1	1	5
In my opinion, the quality of the BESTweb system was below average compared to what exists in industry.			1								1
In my opinion, the quality of the BESTweb system was average compared to what exists in industry.	1			1	1	1		1	1	1	7
In my opinion, the quality of the BESTweb system was above average compared to what exists in industry.		1					1				2
In my opinion, the documentation that came with the BESTweb system was below average compared to what exists in industry.											0
In my opinion, the documentation that came with the BESTweb system was average compared to what exists in industry.	1		1	1	1	1		1	1	1	8
In my opinion, the documentation that came with		1					1				2

	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	Total
the BESTweb system was above average compared to what exists in industry.											
Reported feeling rusty with the Java language.		1	1	1							3
Reported a lot of trouble understanding the *Best Codes Manager* (BCM) –related portion of the system.		1					1				2
Reported only skimming the BESTweb architecture document (instead of thoroughly reading it).	1		1			1	1	1	1		6
Reported having trouble understanding parts of the system due to poor naming of variables or classes.	1		1				1				3
Extensive experience of working with UML			1								1
Found the UML diagrams useful	1	1	1	1	1	1	1	1	1	1	10
Reported that UML was helpful in gaining an overview	1	1	1		1	1	1			1	7
Did not use UML to the maximum extent, reason: did not want to risk wasting the client's time	1	1		1							3
Did not use UML to the maximum extent, reason: not used to working with UML	1	1		1				1			4
At some point was not able gain an understanding of a piece of the system using UML	1	1		1			1	1	1	1	7
Took advantage of the Use Case Diagram	1	1	1		1	1	1	1		1	8
Took advantage of Sequence Diagrams	1	1	1	1	1	1	1	1	1	1	10
Took advantage of Class Diagrams	1	1	1	1			1				5
Took advantage of Page Flow Diagrams	1	1	1					1	1		5
Took advantage of the Statechart Diagram			1								1
Reported problems comprehending the large Sequence Diagram (quantified in Table 5.4)	1				1		1			1	4

	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	Total
Reported problems comprehending the large Class Diagram (quantified in Table 5.5)	1		1						1		3
Felt he completed the tasks faster thanks to the UML		1							1		2
Felt he completed the tasks slower due to the UML					1			1			2
Did not trust the UML diagrams to be accurate								1			1
Reported needing Javadoc comments when looking at the diagrams	1								1	1	3
Found the UML tool to be problematic	1	1	1	1	1	1	1	1	1		9
Did not look at the UML diagrams during development							1	1			2
Used the UML for navigation (browsing the system)	1		1	1	1	1	1		1	1	8
Used the UML diagrams to find exact code change places			1		1	1	1		1	1	6
Found the UML tool training adequate	1	1	1	1	1	1	1	1	1	1	10

Appendix C The Tasks

C.1 Persist Query

Add functionality to remember (persist) the user's last query.

- A user performs a search by typing a query into the *Search Query* textbox and clicking the *Search* button on the *Main Page*. The *Main Page* is the page displayed right after logging in from the *Welcome Page*, this is the first page one sees when first accessing the system.
- Currently, a search query is forgotten as soon as a user's session expires (it is kept in session scope). We would like to change this behavior so that the next time a user logs into the system they should see their last search query in the *Search Query* textbox. The query should be re-executed and the *Main Page* should display the results of this query.
- Note that the *BESTcode Filter Settings* and *Display Settings* are not to be persisted. The default settings for these will be used.
- When logging back into the system, and if the last search performed by the user was an invalid query (a search query that cannot be parsed properly, e.g. `abc(`), the query should be displayed in the *Search Query*, an error message should specify that the query could not be performed, and the all the publications should be displayed.
- Expand the logging to capture:
  - When a query is being saved, for example, in the case of user x:
    - `Saving query for user [x]: hans`
  - When a query is being loaded, for example, in the case of user x:
    - `Loading user [x]'s last query: hans`

C.2 Add an EndNote Field

Add support for the ALTERNATE_TITLE filed of the BESTweb library file (XML).

- The BESTweb system gets its publication data by an administrator uploading an EndNote generated XML library to the system (see EndNote.xsd in the *Architectural Description* document). Currently not all of the fields are being used. When an admin uploads a library to the BESTweb system where one of the publications contains data in an unused filed the user is warned of the fact that even though the field contained data, this data was skipped (since the system isn't set up to handle it).

- We want to modify the system so that the currently unused field `ALTERNATE_TITLE` is recognized by the system. Specifically, we would like to modify the behavior of the system in the following manner:
  - Searching: The data (text) in that field is searchable in the same manner as for the existing `Title` field data: anywhere in the publication or in a specific field. For example if you search for the query `probability AND assessors` you will get two results, yet only one of the hits has the words portability and assessors in the title. This is because the scope of that search query is not limited to the title data. On the other hand, if you are only interested in searching for publications that have those two words only in the title, you would form you search query like this: `Title:(probability AND assessors)`. You must duplicate this behavior for the `ALTERNATE_TITLE` where the keyword in the search query to limit the scope to the data in that field will be `AlternateTitle`.
  - Publication Details: When a user clicks on a publication displayed on the *Main Page* a *Publication Details* pages comes up. On that page all the data relevant to that specific publication is displayed. Add the *Alternate Title* information to the *Detail Publication Data* of that page. Note that if a publication does not have an alternate title, a blank (nothing) is shown.
  - Display Settings: Modify the *Publications List Display Settings* page so that it is possible to select *Alternate Title*. The *Main Page* being displayed with a column entitled *Alternate Title* where the alternate title for each publication is shown. If an application does not have an alternate title then a blank (nothing) is shown.
  - Main Page when the Alternate Title is selected in the Display Settings: If the *Alternate Title* header is clicked publication list must be sorted with respect to each publication's *Alternate Title*. To indicate that the list is being sorted according to this field, an arrow is to appear by the header's (*Alternate Title*) title. If the sorting is in the alphabetical order the arrow must point up. Conversely, the arrow must point down if the order is reverse-alphabetical. The sorting order is toggled by the same header being clicked subsequently.

C.3 BEST Codes Management: Add Codes & Categories

BEST category and codes management: addition of categories & codes.

- Present the user with the options to (and add functionality to):
  - Add a BEST code.

- Add a BEST code category.
- Update the system using the new set of BEST codes & categories.

- In the case of addition of a BEST code:
  - Present the user with the option to choose an existing category (category IDs and their descriptions) they wish to add the new BEST code to.
  - A textbox must be present for the name of the new code.
  - A code is valid if:
    - It is exactly two characters long.
    - The first character is alphabetical and upper case, i.e., A to Z.
    - The second character must be alphabetical and lowercase, i.e., a to z.
    - E.g.: Ab and Bb are valid codes, while AZ, de, A9, and 9f are not.
    - The code name must be unique, different from already-existing code names.
  - A textbox must be presented for the description of the code.
    - A user must enter a description for the code.
  - If the user fails to meet any of the requirements, an error message must be presented to the user specifying the requirement they did not meet. E.g.: "You did not enter a description for your new code."
  - Upon a successful addition of the code the user must see a message which specifies that the code was added successfully.
- In the case of addition of a BEST code category:
  - A textbox must be present for the description of the new category.
    - The description of the category must be unique from the description of the other categories. If this description is not unique the user must be notified of this and asked to modify the description.
  - The ID (number) of the category must be assigned automatically. This number must equal the largest existing category ID number + 1.
  - Upon a successful addition of the category the user must see a message which specifies that the category was added successfully.
- Once the user is finished adding the BEST codes & categories they must be given the option to "update the publications" so if any of the existing publications contained these new BEST codes (that the system previously skipped because it didn't recognize them) can now be added to the publications.
- The user account must have administrator privileges to use this functionality.

- You can assume that only one administrator will be making changes to the BEST codes at a time.
- Keep the GUI simple, like the GUI for the other admin tasks.
- Expand the logging to capture information about:
  - When a code/category is being added to the system, for example:
    - `[x] added category: 6-Some Description`
  - Unsuccessful attempts at adding a code/category added to the system, for example:
    - `[x] tried to add an existent BEST code to the system: 6-Be`

C.4 Cache Default Graphs

Add caching of graphical statistic results for the complete publications list.

- Currently when the user views a statistics graph (accessed by clicking the *View Statistics* link off of the *Main Page*) the graph is generated based on the publications that the user sees on the *Main Page*.
- The users will frequently want to see statistics for all the publications in the system. Since we expect this to happen often we want the statistics data for all publications to be cached so that it does not have to be regenerated for every user.
- Please modify the system so that the data used by the statistical graphs for the complete publication list does not have to be regenerated for every user. Rather, this data should only be generated at system startup and when the cached data becomes invalid (inconsistent with what would be shown if caching was not used).
- Expand the logging to capture information about:
  - When a cached data is being generated, for example:
    - `Generating cached data for: Publications per year`
  - When cached data is being accessed, for example:
    - `[x] is viewing cached data for: Publications per year`

C.5 BEST Codes Management: Delete Codes & Categories

BEST category and codes management: deletion of categories & codes.

- Present the user with the options to (and add functionality to):
  - Remove a BEST code.
  - Remove a BEST code category.
  - Update the system using the new set of BEST codes & categories.
- In the case of deletion of a BEST code:
  - Present the user with a list of the existing categories.

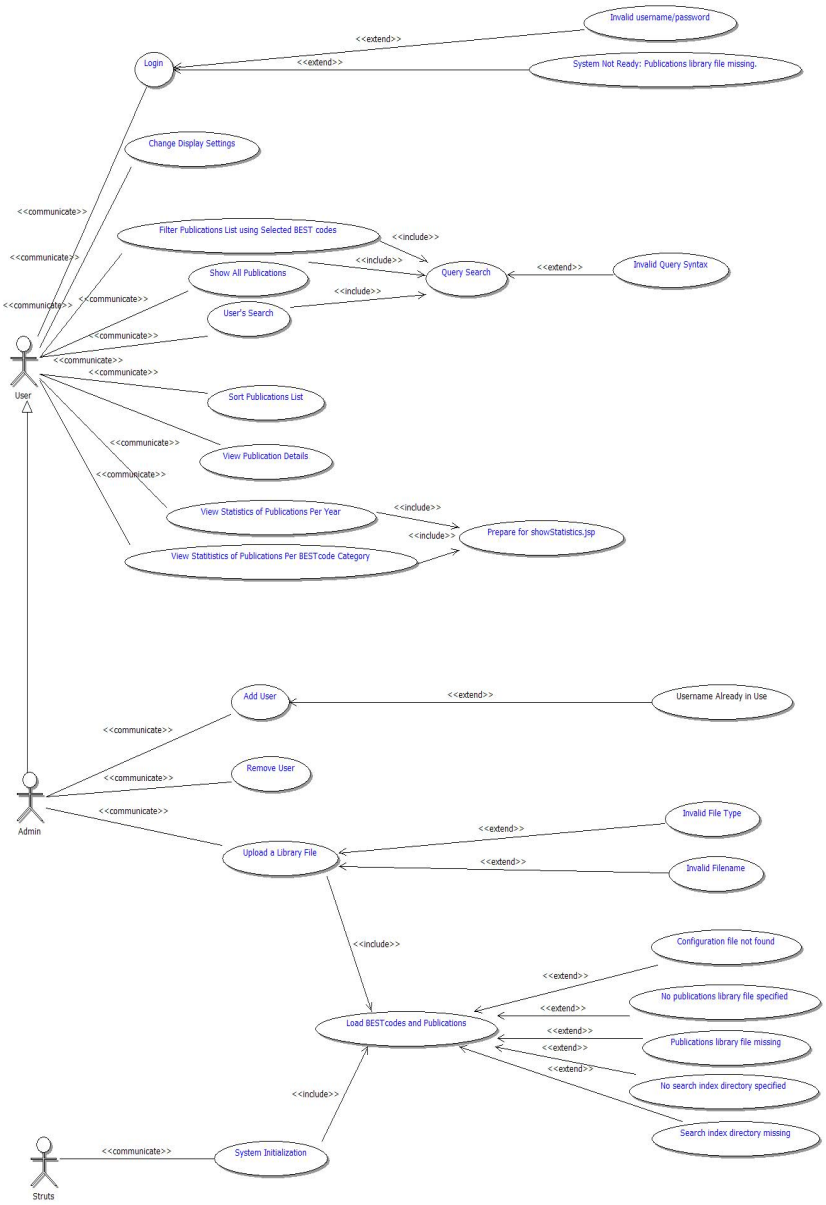
- Once the user selects a category, present the user with a list of the codes in the selected category.
- The user will click on the code they wish to remove.
- Present the user with a message that the code has been successfully deleted.

- In the case of deletion of a BEST code category:
  - Present the user with a list of the existing categories.
  - Once a user selects a category they wish to delete, display a warning on the page that when a category is deleted the corresponding BEST codes will also be removed (corresponding to that category).
  - Present the user with a message that the category has been successfully deleted.
- Once the user is finished deleting the BEST codes & categories they must be given the option to "update the publications" so that the deleted BEST codes and categories are no-longer associated with the publications.
- The user account must have administrator privileges to use this functionality.
- You can assume that only one administrator will be making changes to the BEST codes at a time.
- Keep the GUI simple, like the GUI for the other admin tasks.
- Expand the logging to capture information about:
  - When a code/category is removed from the system, for example:
    - `[x] removed category: 6-Some Description`

Appendix D UML Documents

D.1 Use Cases

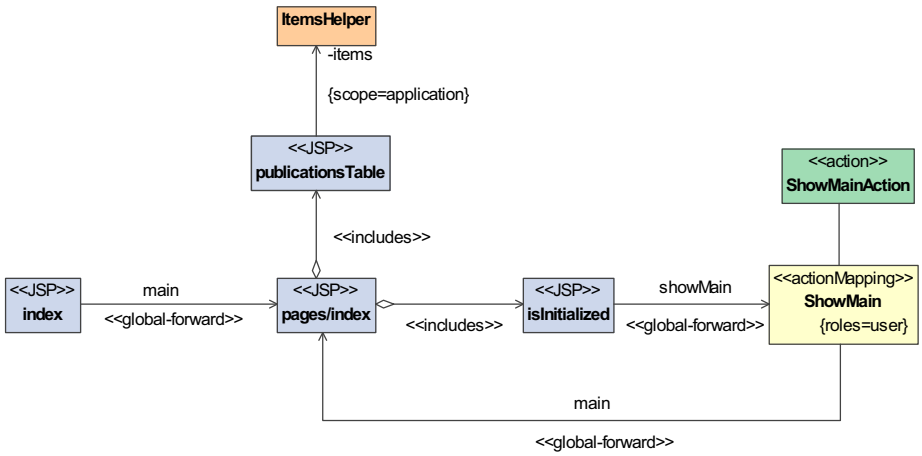
D.1.1 Use Case Summary

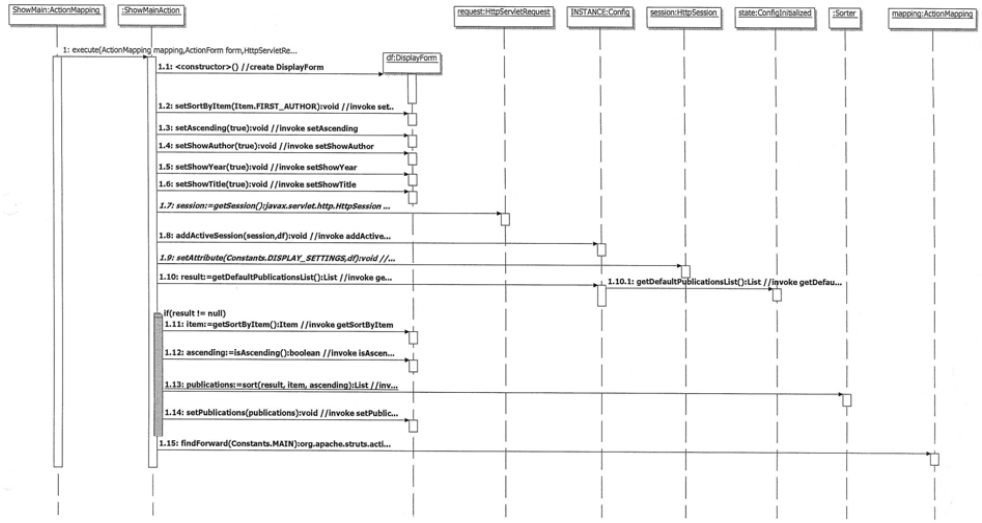


Login	The `User` gains access to the system and is forwarded to the Main Page.
Change Display Settings	The `User` changes the details that are shown on the main page about the publications. E.g. Show all the authors be shown vs. just the first author.
Filter Publications List using Selected BESTcodes	The `User` refines the publications list by limiting the list to only display the publications with at least one of the selected BESTcode(s).
Show All Publications	The `User` chooses to see all the publications in the system implicitly clearing the search query and resetting the BESTcode filtering settings.
User's Search	The `User` searcher for a query. The publications list shows the results of the query. If there are zero matches, a message is displayed in place of the publications list notifying the `User` of this.
Sort Publications List	The `User` alters the way in the publications list is sorted. E.g. By year in stead of by the first author's name.
View Publication Details	The `User` views all the information related to a specific publication (identified by a unique ID number).
View Statistics of Publications Per Year	The `User` views statistics based on the publications list on the main page: the number of publications per year.
View Statistics of Publications Per BESTcode Category	The `User` views statistics based on the publications list on the main page: the number of publications per BESTcode in a specific (specified) BESTcode-category.
Add User	The `Administrator` adds a new user of the BESTweb system to the system with a unique username.
Remove User	The `Administrator` deletes one of the BESTweb users from the system.
Upload a Library File	The `Administrator` uploads a new BESTweb publications library file to the system.

D.1.2 Login

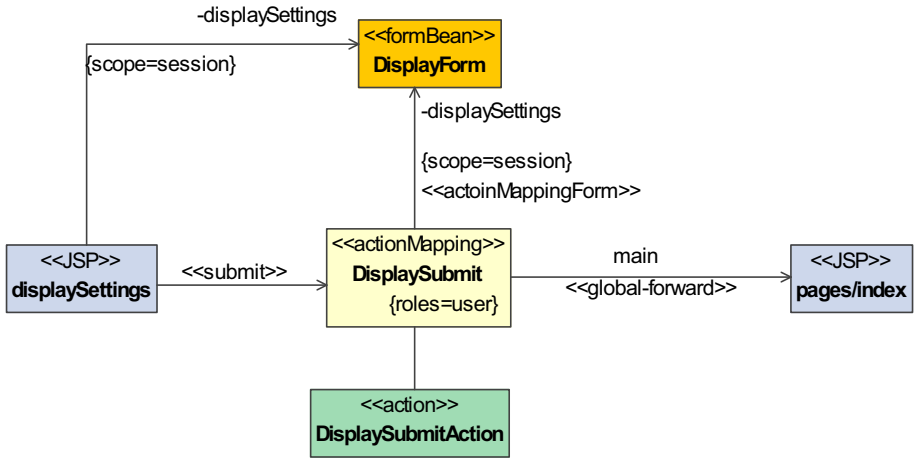
Name	`Login`
Participating actors	Initiated by `User`.
Flow of events	1. The user tries to access the main page by clicking on the "Enter the BESTweb system" link. 2. The system prompts the user for his username and password. 3. The user enters his username and password and clicks OK. 4. The system verifies that the username and password are correct and, if they are, grants access to the page. 5. The system sets the user 's display settings to the default: • Show first author's name, the publication's year, the title. • Show the publications list sorted by the first author's name in the ascending order. 6. The system sets the user's publication list to the default list (all the publications in the system).
Exit conditions	• The `User` is logged into the system. • The user's publications list is set to the default list (all publications). • The main page shows all the publications in the system. • The user's display options are set to the default. • The publications list is sorted according to the user's preferences.

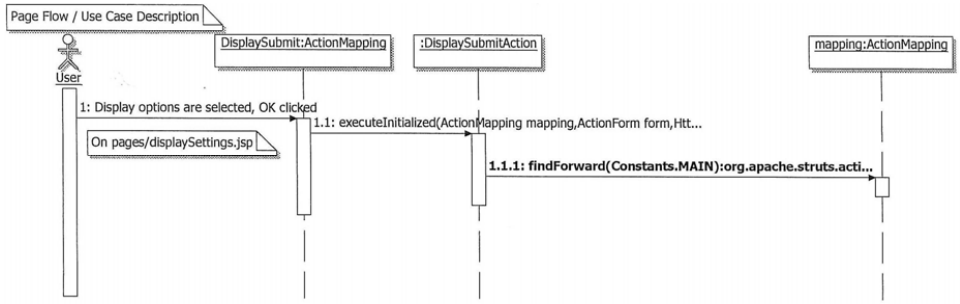




D.1.3 Change Display Settings

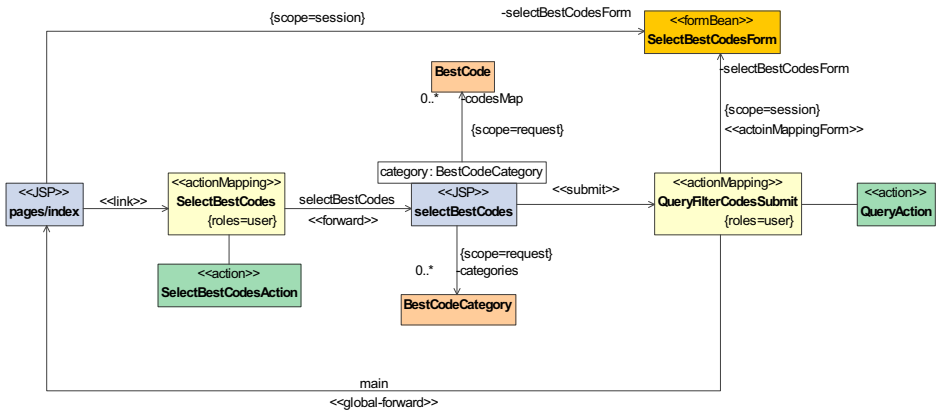
Name	`Change Display Settings`
Participating actors	Initiated by `User`.
Flow of events	1. The user chooses the "Display Settings" link from the main page. 2. The system forwards the user to the "Publications List Display Settings" page where the display options are shown (currently selected options are checked). 3. The user checks/unchecks the desired options and presses the OK button. 4. The system saves the updated options and forwards the user to the main page. 5. The main page is re-generated using the new display options.
Entry condition	The `User` is logged into the system.
Exit conditions	• The main page only shows the items of the publications specified in the user's new display settings.

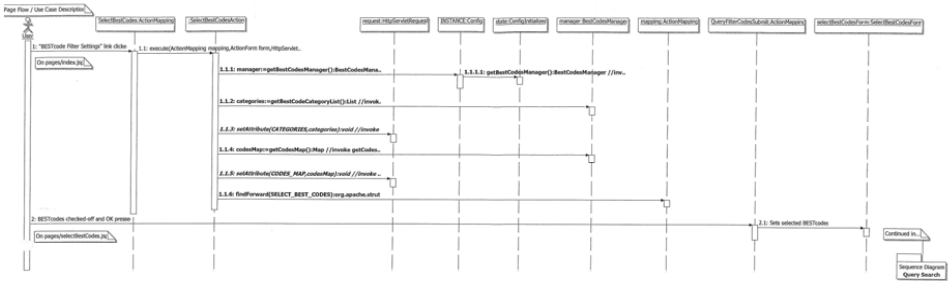

-displaySettings
{scope=session}
<<formBean>>
DisplayForm
-displaySettings
{scope=session}
<<actoinMappingForm>>
<<JSP>>
displaySettings
<<submit>>
<<actionMapping>>
DisplaySubmit
{roles=user}
main
<<global-forward>>
<<JSP>>
pages/index
<<action>>
DisplaySubmitAction


Page Flow / Use Case Description
User
DisplaySubmit:ActionMapping
:DisplaySubmitAction
mapping:ActionMapping
1: Display options are selected, OK clicked
On pages/displaySettings.jsp
1.1: executeInitialized(ActionMapping mapping,ActionForm form,Htt...
1.1.1: findForward(Constants.MAIN):org.apache.struts.acti...

D.1.4 Filter Publications List using Selected BESTcodes

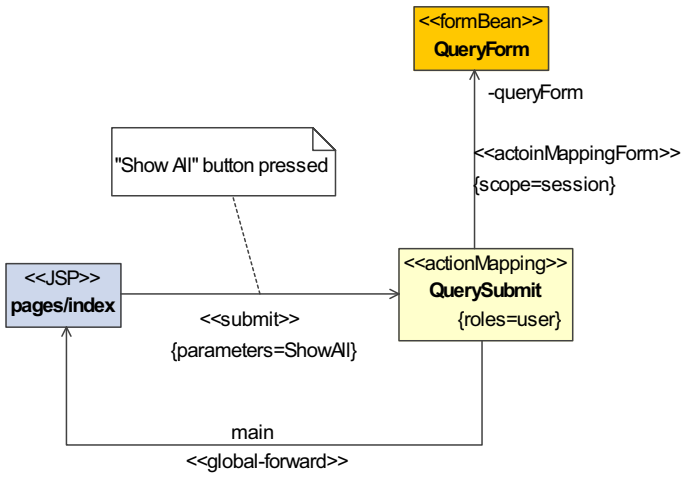
Name	`Filter Publications List using Selected BEST codes`
Participating actors	Initiated by `User`.
Flow of events	1. The user chooses the "BESTcode Filter Settings" link from the main page. 2. The system forwards the user to the "BESTcodes Search Filter Settings" page where all the BESTcodes are shown (currently selected BESTcodes are checked). 3. The user checks/unchecks the BESTcodes that he is interested and presses the OK button. 4. The system saves the selected BESTcodes. 5. The system filters the user's publications list so that only publications that have at least one of the selected BESTcodes are shown. If there a search query is present, only the publications that contain the search query are shown. This takes place in the included Use Case: Query Search.
Entry condition	The `User` is logged into the system.
Exit conditions	• The main page only shows publications that: (a) match the user's search query (if any), and (b) contain at least one of the BESTcodes selected by the user. • The selected BESTcodes are specified on the main page.

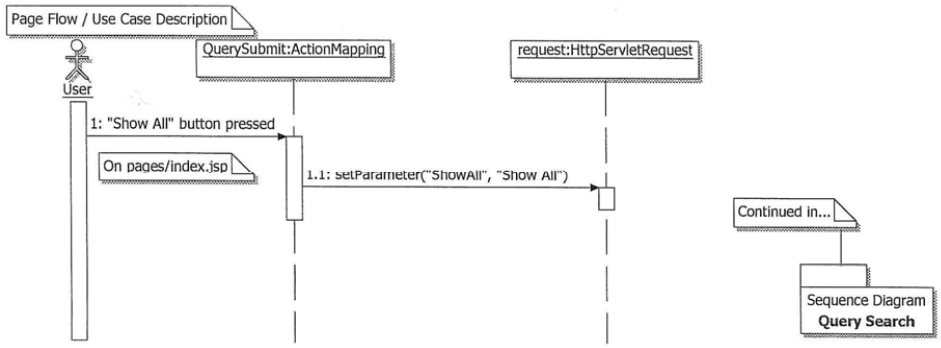




D.1.5 Show All Publications

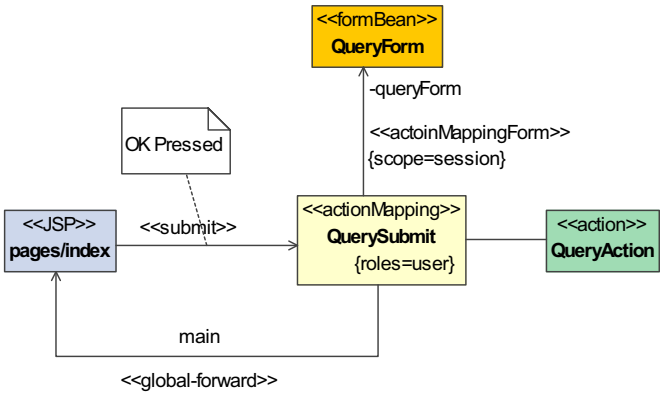
Name	`Show All Publications`
Participating actors	Initiated by `User`.
Flow of events	1. The user chooses the "Show All" button from the main page. 2. The systems clears the user's query . 3.The systems clears the user's BESTcode filter settings. 4. The system sets the user's publications list to the default list (all publications). This takes place in the included Use Case: Query Search.
Entry condition	The `User` is logged into the system.
Exit conditions	• The main page shows all the publications. • The search query is blank. • BESTcodes filter settings are cleared.

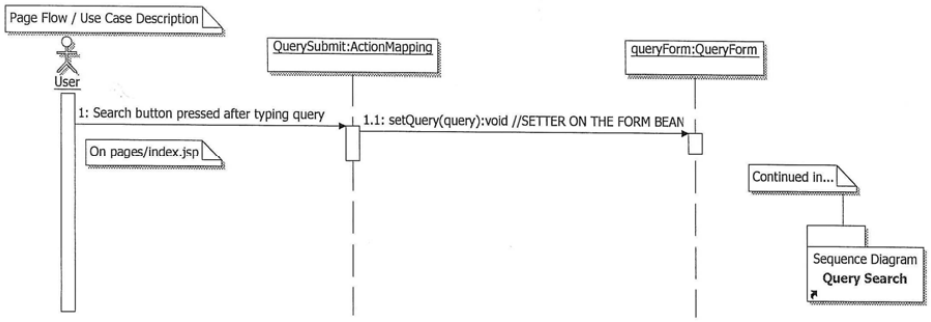




D.1.6 User's Search

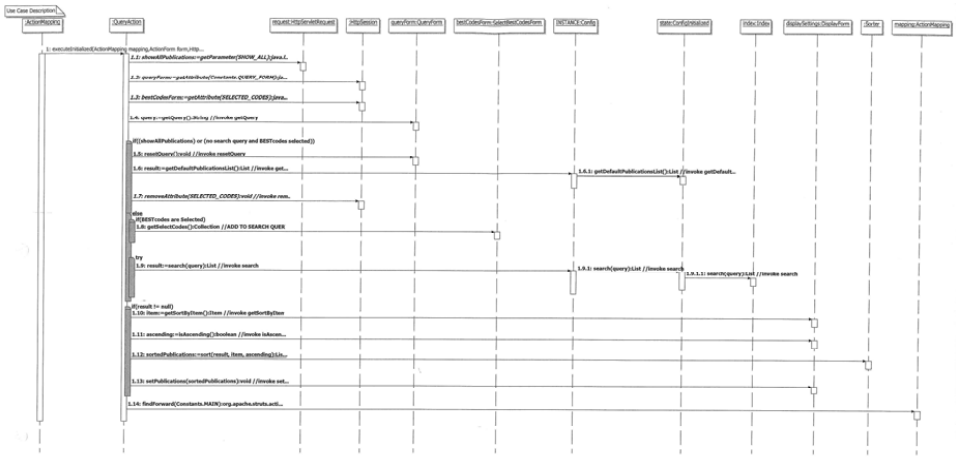
Name	`User's Search`
Participating actors	Initiated by `User`.
Flow of events	1. The user types in a query in the "Search Query" box and presses the "Search" button on the main page. 2. The system filters the user's publications list so that only publications that contain the search query are shown.If BESTcodes are selected, the show publications must also have at least one of the selected BESTcodes. This takes place in the included Use Case: Query Search.
Entry condition	The `User` is logged into the system.
Exit conditions	• The main page only shows publications that: (a) match the user's search query (if any), and (b) contain at least one of the BESTcodes selected by the user.





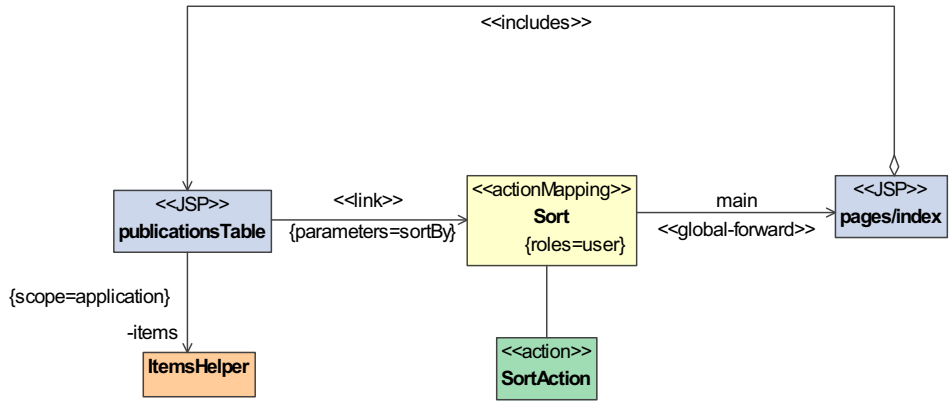
D.1.7 Query Search

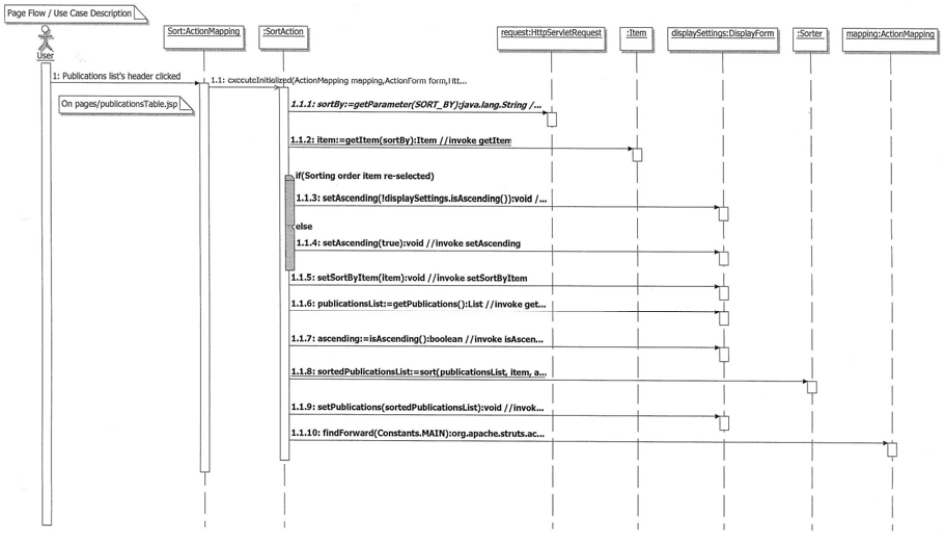
Name	Query Search
Flow of events	1. If the user wants to see all the publications or search query is blank and there are no BESTcode filter options set, the user is assigned the default (complete) publications list. A request to see all the publications (Show All) clears the search query. 2. If there is a search query, the list of publications is limited to the publications that contain the search query. 3. If any BESTcode filters are set, the list of publications is further limited (in the case that a query was specified) to only the publications that contain at least one of the selected BESTcodes. 4. The publications list is sorted using the user's sort settings. 5. The system forwards the user to the main page where which is re-generated using the new publications list.



D.1.8 Sort Publications List

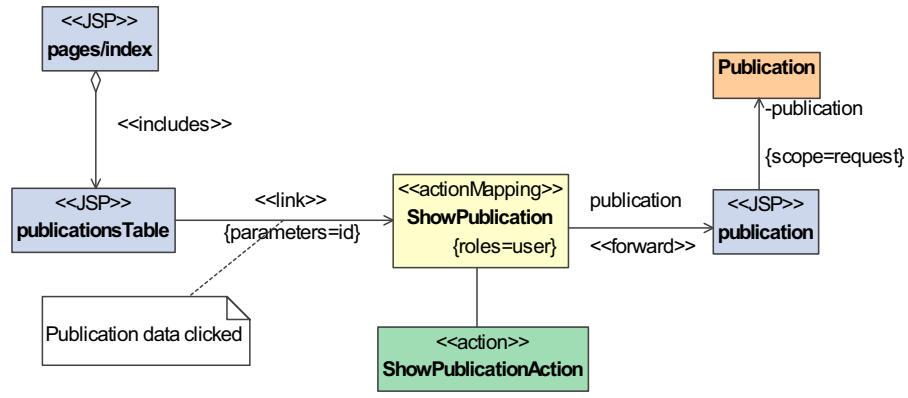
Name	`Sort Publications List`
Participating actors	Initiated by `User`.
Flow of events	1. The user clicks on one of the publications list's headers on the main page. 2. The system sorts the user's publications list according the clicked header. 3. If the same header is repeatedly clicked, the sorting is altered between ascending and descending. 4. The user's display settings are updated.
Entry condition	The `User` is logged into the system.
Exit conditions	• The publications list is sorted according to the selected header. • The user's display settings are updated.

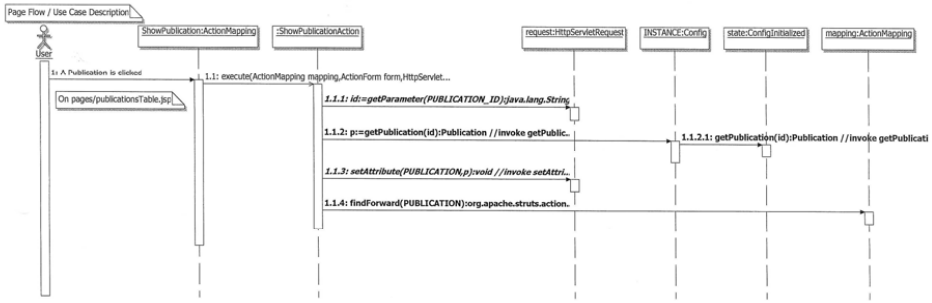




D.1.9 View Publication Details

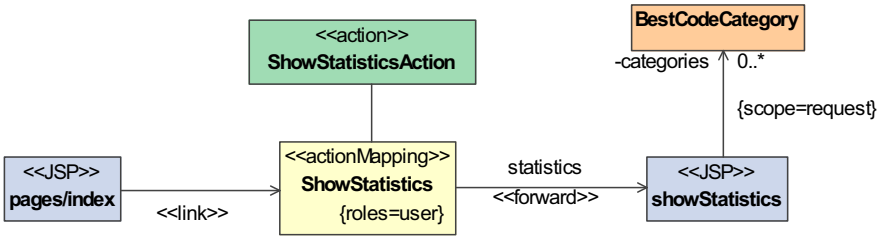
Name	`View Publication Details`
Participating actors	Initiated by `User`.
Flow of events	1. The user clicks on one of the publications in the publication's list. 2. The system shows the details of the selected publication in the "Publication Details" page.
Entry condition	The `User` is logged into the system.
Exit conditions	• The details of the selected publication are shown in the "Publication Details" page

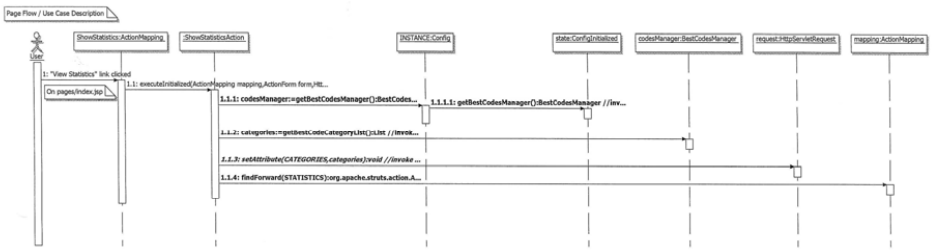




D.1.10 Prepare for showStatistics.jsp

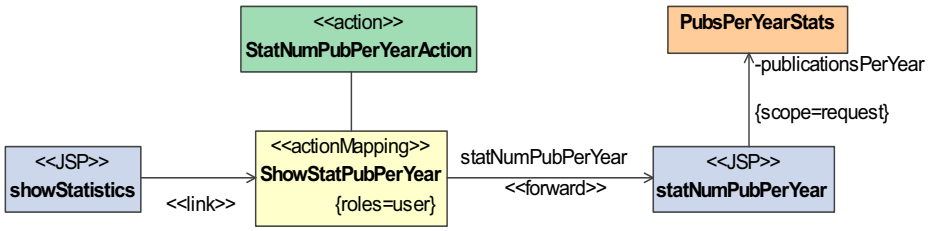
Name	`Prepare for showStatistics.jsp`
Flow of events	1. The system configures the "Show Statistics" page so that, for each BESTcode category, the user can click the following link: • Number of publications per BESTcode for category: X - category_description Where X is the category ID and category_description is the description of that category.

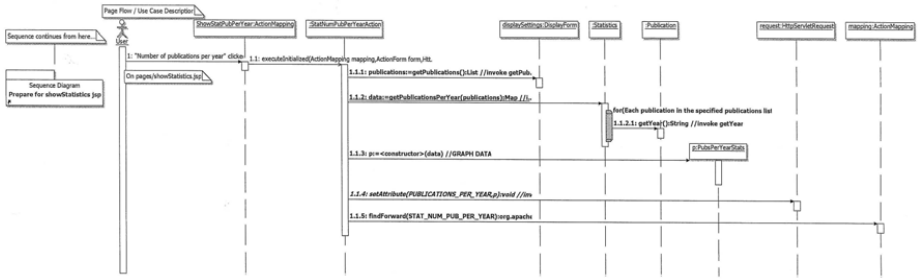




D.1.11 View Statistics of Publications Per Year

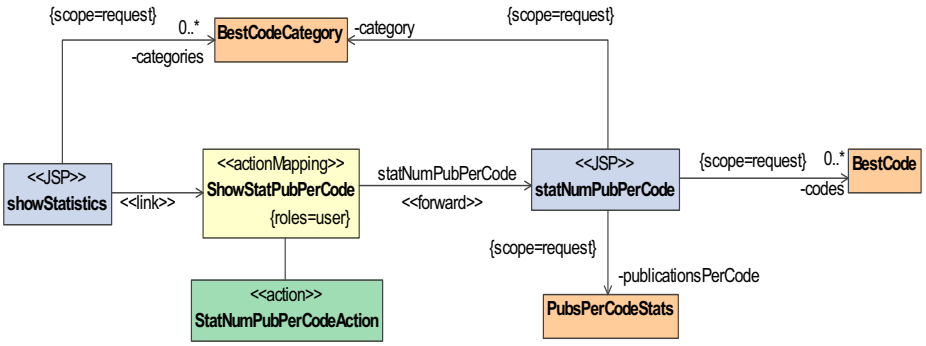
Name	`View Statistics of Publications Per Year`
Participating actors	Initiated by `User`.
Flow of events	1. The user clicks on the "Number of publications per year" link on the "Show Statistics" page (pages/showStatistics.jsp). 2. The system displays a graph with the publication years on the X axis and the number of publications in that year on the Y axis. Only years for which there is at least one publication are shown. The graph is based on the user's publications list.
Entry condition	The `User` is logged into the system.
Exit conditions	• A statistics graph showing the number of publications per year, based on the user's publications list, is shown to the user. The publication years on the X axis and the number of publications in that year on the Y axis. Years with no publications are omitted.

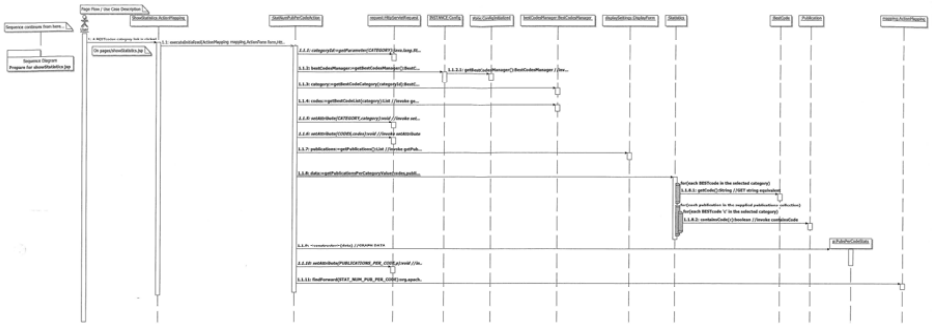




D.1.12 View Statistics of Publications Per BESTcode Category

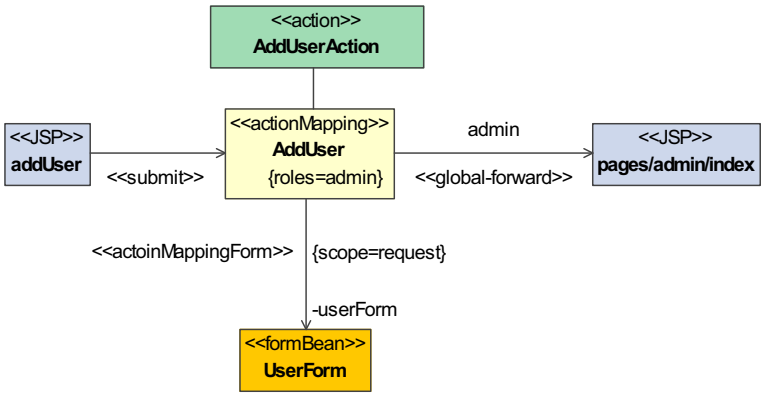
Name	`View Statistics of Publications Per BESTcode Category`
Participating actors	Initiated by `User`.
Flow of events	1. The user clicks on the "View Statistics" link on the main page. 2. The system executes the included Use Case: "Prepare for showStatistics.jsp". 3. The user clicks on the BESTcode category they wish to see statistics for. 4. The system generates a graph which displays the number of publications (Y axis) for each BESTcode in the selected category (X axis). The system also displays a legend that gives the description for each BESTcode. The graph is based on the user's publications list.
Entry condition	The `User` is logged into the system.
Exit conditions	• A statistics graph showing the number of publications per BESTcode in a specific BESTcode-category, based on the user's publications list, is shown to the user. The BESTcodes are on the X axis and the number of publications that have that best BESTcode on the Y axis.The graph is based on the user's publications list. All the BESTcodes in the category are shown. • A legend of the BESTcodes (containing the description of the BESTcodes) is also displayed.

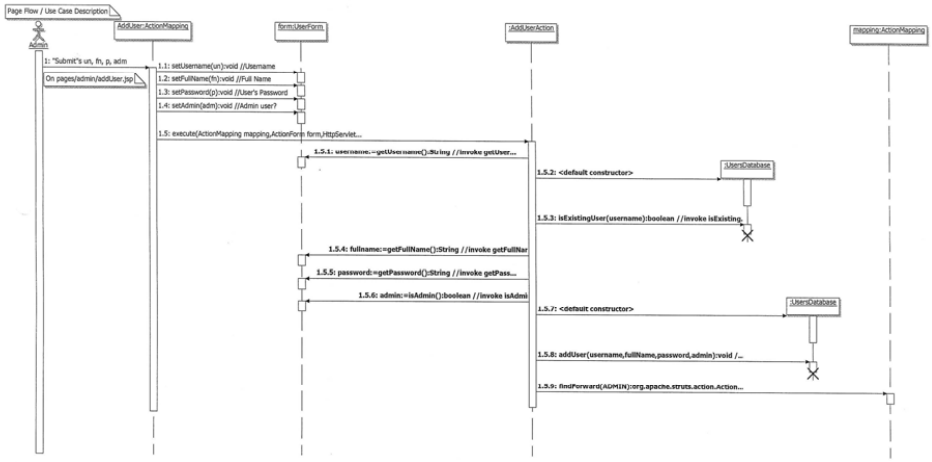




D.1.13 Add User

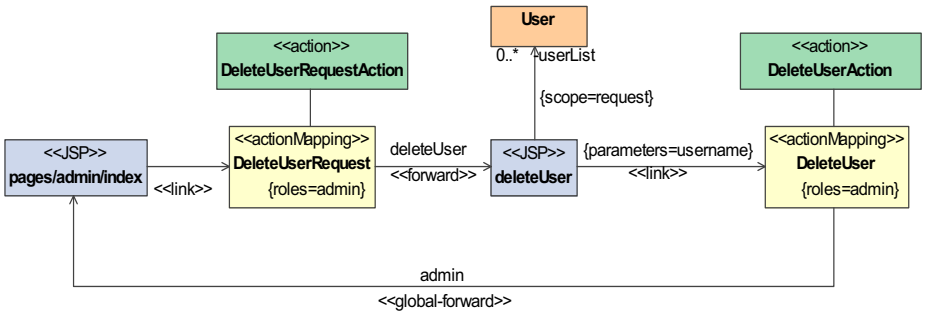
Name	`Add User`
Participating actors	Initiated by `Admin`.
Flow of events	1. The admin enters the "Add User" page (pages/admin/addUser.jsp) 2. The admin enters the new user's username, full name, password, specifies if this user will have admin privileges or not, and presses the Submit button. 3. The system checks if this username is already in use or not. If not, the new user is added. 4. The system brings the admin back to the "Administration Menu" page.
Entry condition	And `Admin` user is logged into the system.
Exit conditions	• A new user is added to the system.

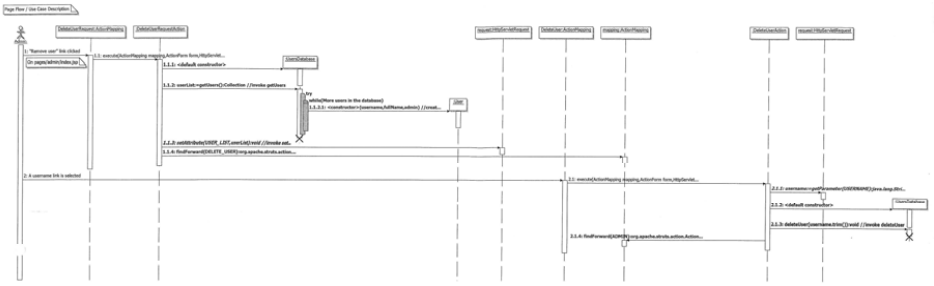




D.1.14 Remove User

Name	`Remove User`
Participating actors	Initiated by `Admin`.
Flow of events	1. The admin selects the "Remove User" link from the "Administration Menu" page. 2. The system presents a list of usernames in the system, each in the form of a link, on the "Delete User" page (pages/admin/deleteUser.jsp). 3. The admin clicks on link corresponding to the user they wish to delete from the system. 4. The system deletes the specified user. 5. The user is forwarded back to the "Administration Menu" page.
Entry condition	And `Admin` user is logged into the system.
Exit conditions	• A user is removed from the system.





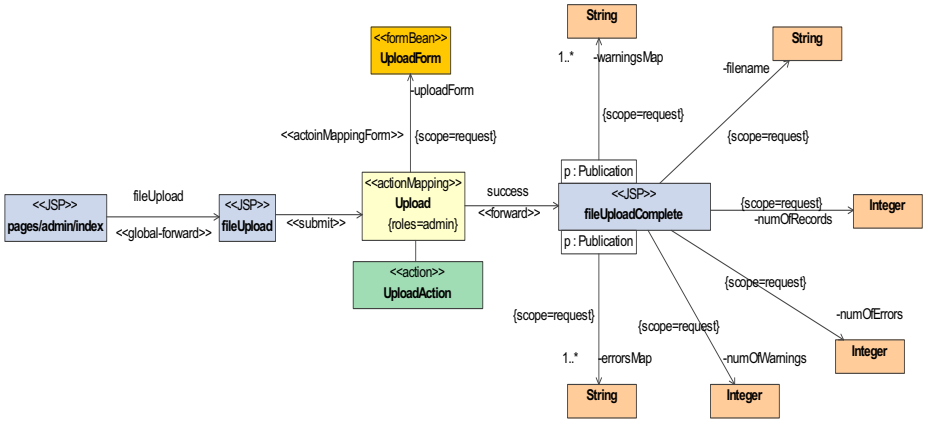
D.1.15 Upload a Library File

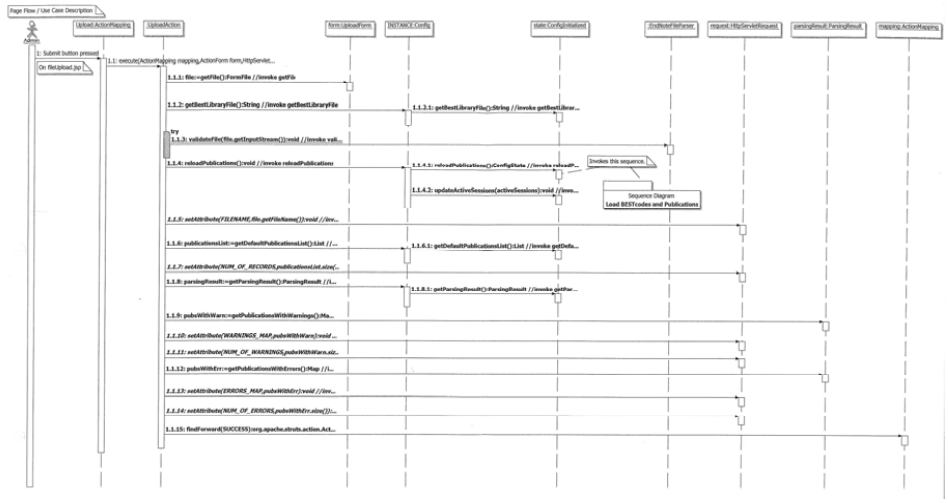
Name	`Upload a Library File`
Participating actors	Initiated by `Admin`.

Flow of events	1. The admin enter the "File Upload" page (pages/admin/fileUpload.jsp) page.
	2. The admin specifies the local file (publications library file in the XML format) they wish to upload and presses the Submit button.
	3. The system ensures that the filename is valid.
	4. The system reads the file.
	5. The system ensures that the file is a valid XML publications library file.
	6. The system parses the file and extracts the publications. This occurs in the included use case: "Load BESTcodes and Publications".
	7. The system replaces all logged-in users' publications lists are with the new default (complete) publications list.
	8. The system forwards the admin to the "Upload Completed" page (pages/admin/fileUploadComplete.jsp).
	9. On that page, the system displays:
	• The number of publications that:
	  ○ Were successfully read from the library file.
	  ○ Contained warnings.
	  ○ Contained errors (could not be imported into the system).
	• For each publication that had warnings, the warnings are displayed.
	• For each publication that had errors, the errors are displayed.

Entry condition	And `Admin` user is logged into the system.

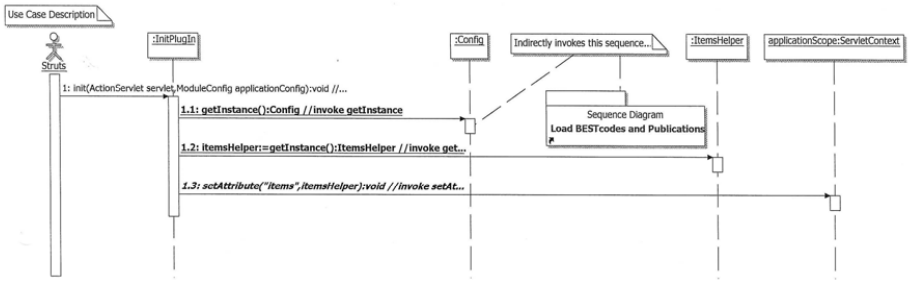
Exit conditions	• A new publications library is uploaded to the system.
	  ○ Any errors/warnings in this publications library are shown the the admin.
	• The system is updated with the publications from this library.
	  ○ The default publications list is updated.
	  ○ All logged-in users' publications lists are replaced with the new default (complete) publications list.





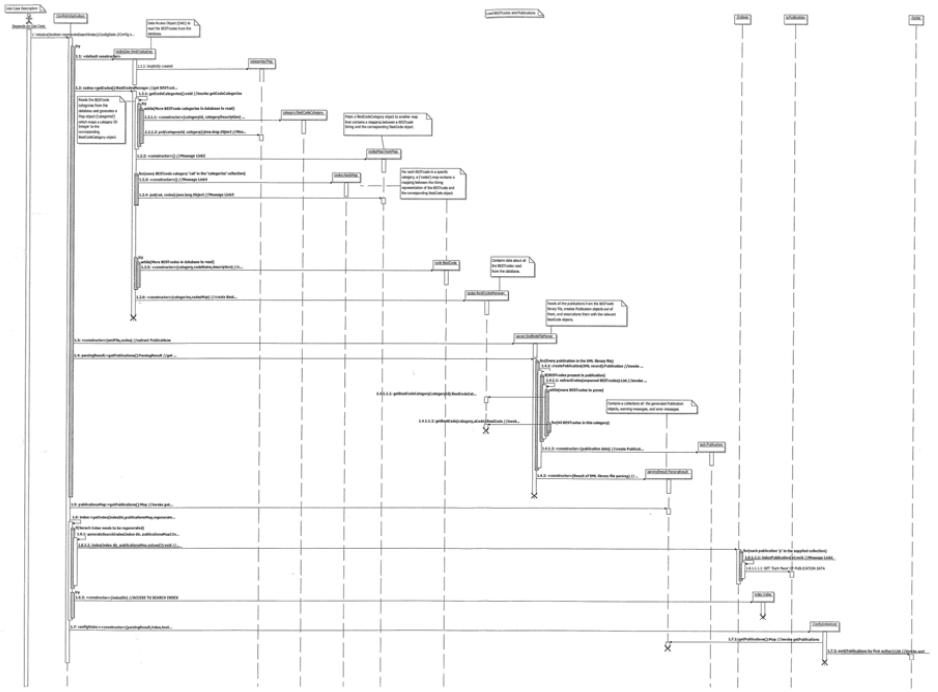
D.1.16 System Initialization

Name	`System Initialization`
Participating actors	Initiated by `Struts`.
Flow of events	1. The system reads the BESTcode-categories, BESTcodes, and the publications library file. This occurs in the included use case: "Load BESTcodes and Publications". 2. An instance of the ItemsHelper class is loaded into application scope.
Exit conditions	• The system is ready for users to log-in.



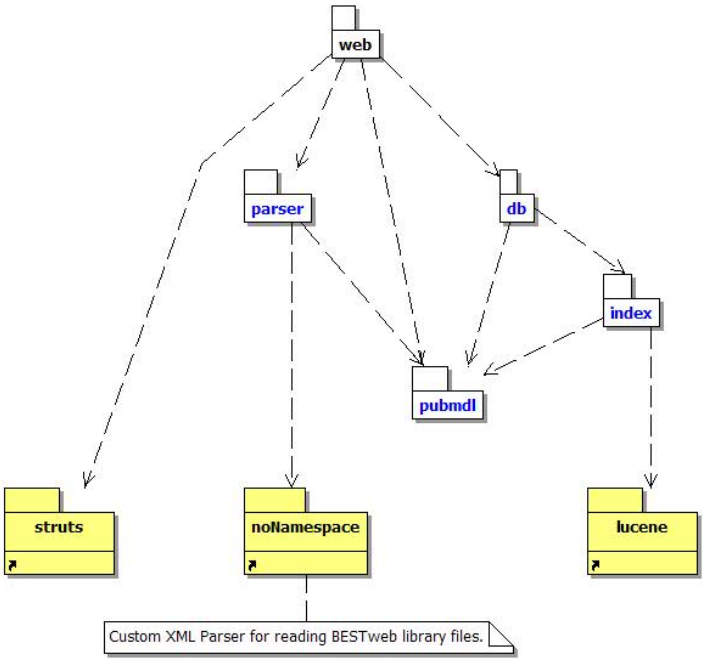
D.1.17 Load BESTcodes and Publications

Name	`Load BESTcodes and Publications`
Flow of events	1. The system builds a BESTcode-categories and BESTcodes model by querying database. 2. The system extracts the publications from the BESTweb publications library file. Each publication is tagged with the relevant BESTcodes (this information is specified in the publications library file). 3. If this is a new library file then a search index is built based on these publications. 4. The system is updated with a new default publications list, sorted in the default manner (by first author, ascending).

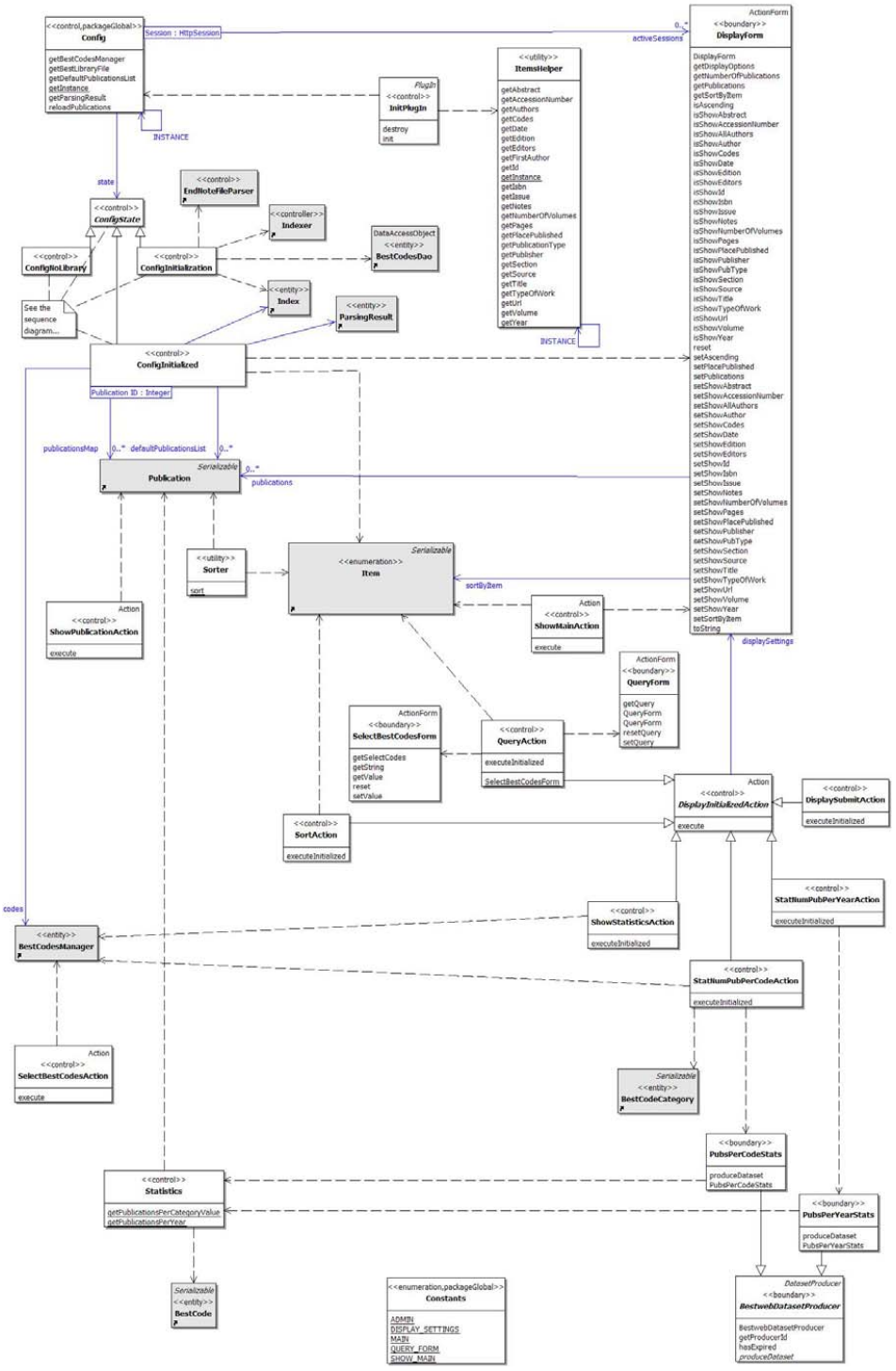


D.2 Class Diagrams

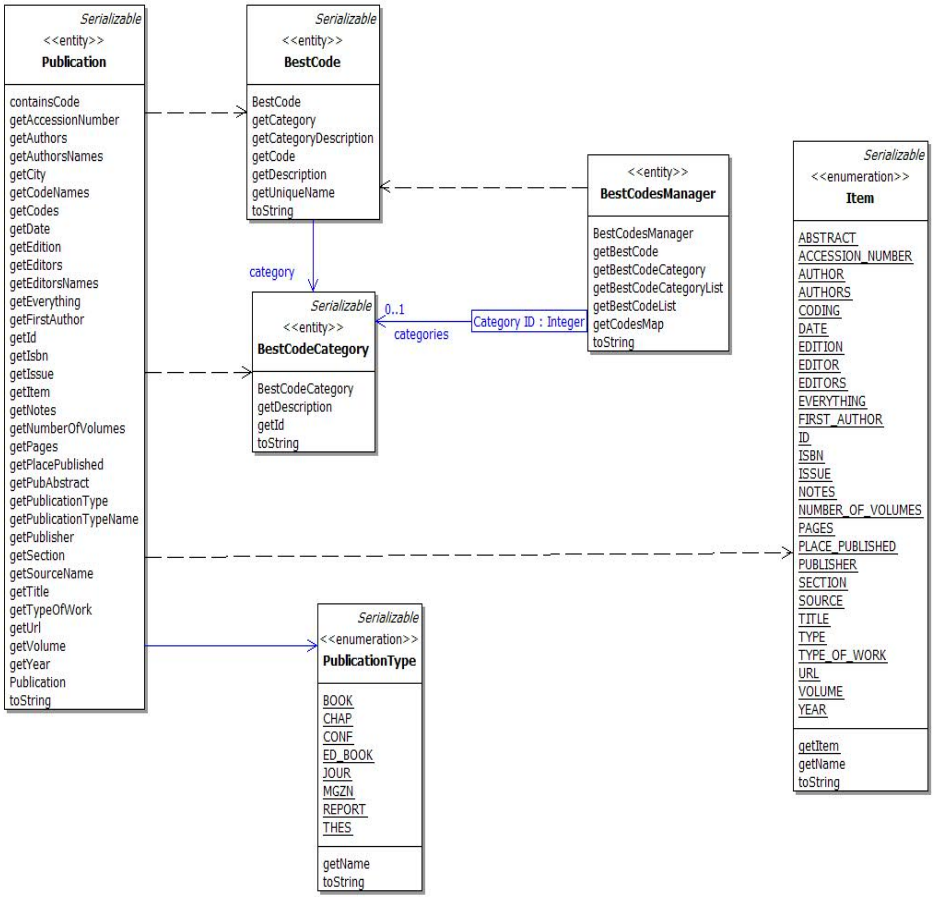
D.2.1 Package Overview



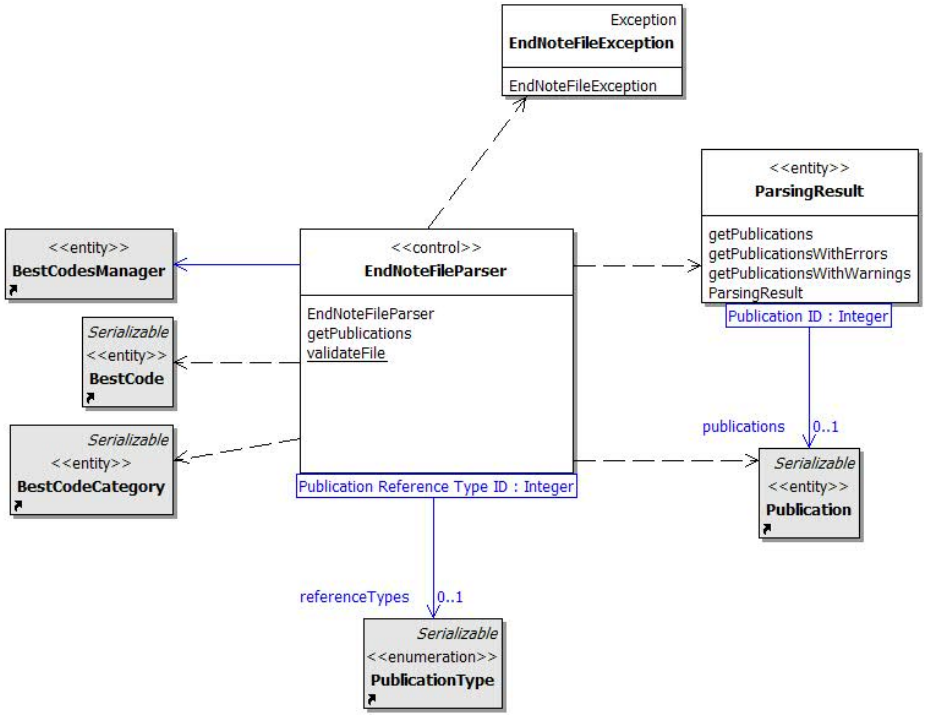
D.2.2 no.simula.bestweb.web



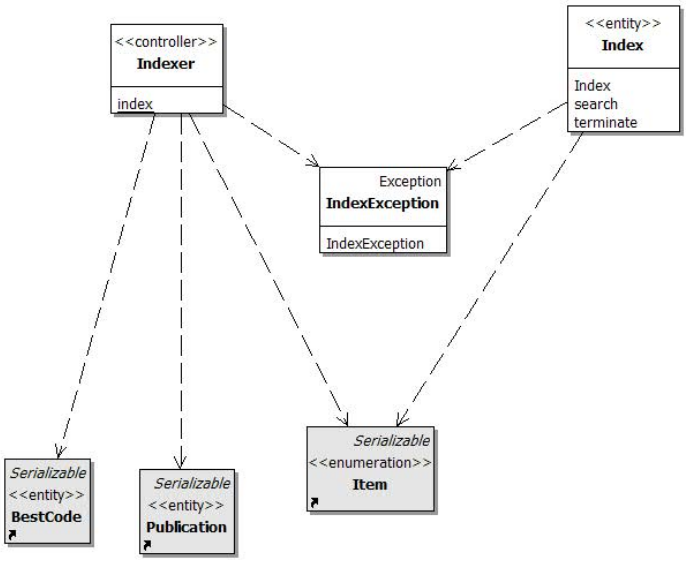
D.2.3 no.simula.bestweb.pubmdl



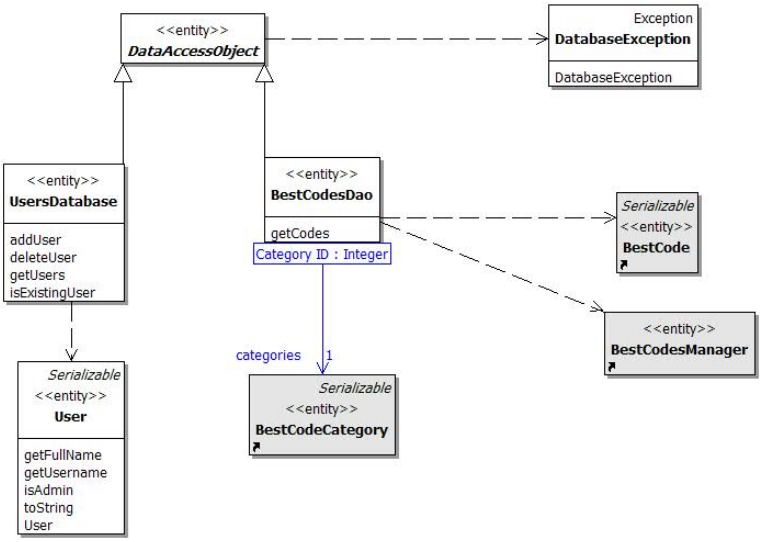
D.2.4 no.simula.bestweb.parser



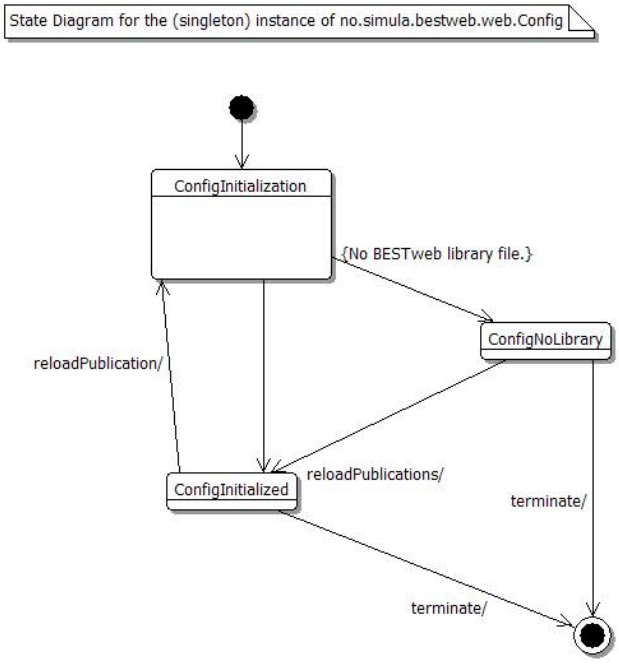
D.2.5 no.simula.bestweb.index



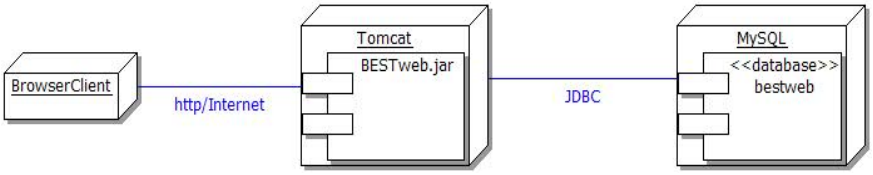
D.2.6 no.simula.bestweb.db



D.3 State Diagram for no.simula.bestweb.web.Config

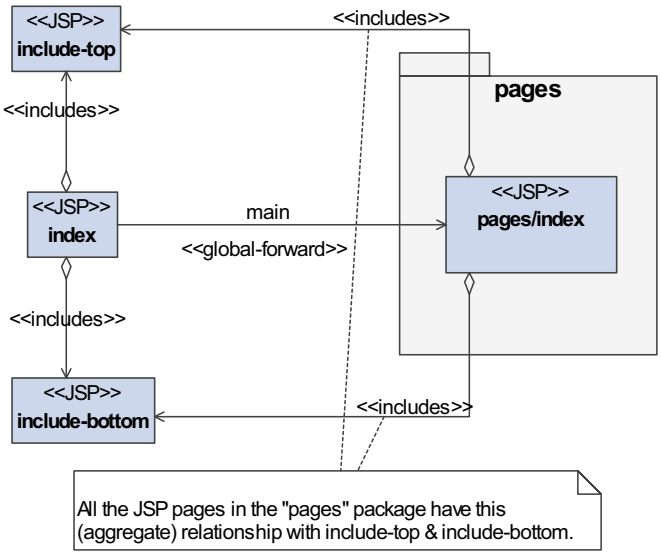


D.4 Deployment Diagram

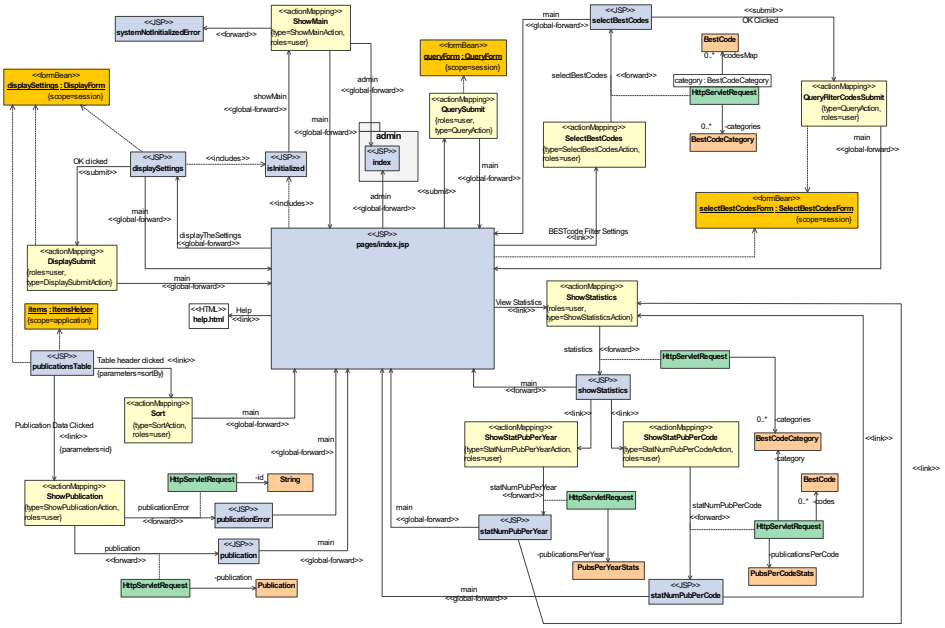


D.5 Page Flow

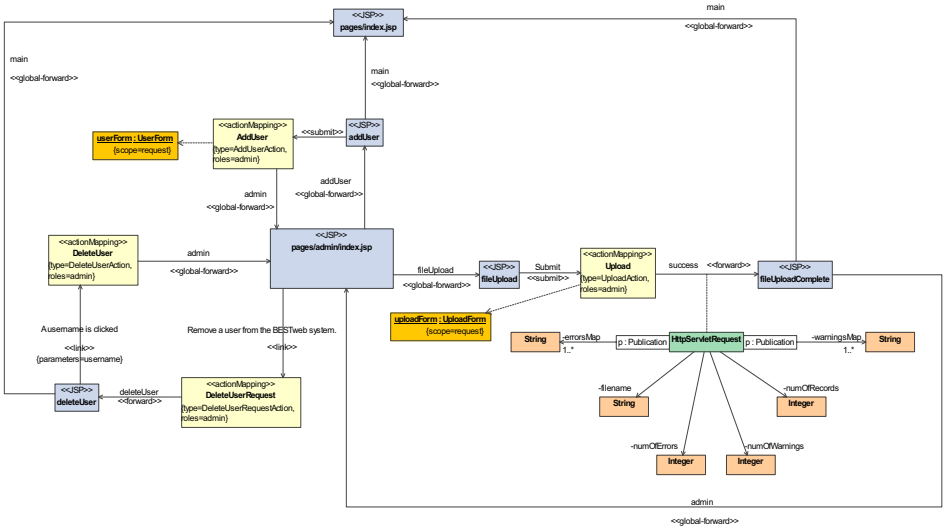
D.5.1 / (base)



D.5.2 Pages



D.5.3 pages.admin



Appendix E BESTweb Architecture Document

This document gives an overview of the BESTweb system from both a functional and a technical perspective.

E.1 Introduction

The BESTweb system is essentially a web-interface to an EndNote (www.endnote.com) generated file (in XML format) containing a collection of data on publications. A publication is, for example, a paper that was submitted to a conference. This paper can have many pieces of data associated with it: the name of the conference that it was submitted to, the year of the conference, etc. The BESTweb system creates a front-end for this file allowing users to, for example, search for publication by a specific author and view statistical data on these publications. Furthermore, most of the publications in the system are categorized according to specific codes: the BEST-codes. The system also takes into account this need for coding and categorization of the publications. Finally, the system also contains an administration interface that allows for uploading of a new EndNote generated file (a BEST-codes library file) and the ability to add and remove BESTweb users.

A more in-depth look at the functionality provided by the system will now be provided.

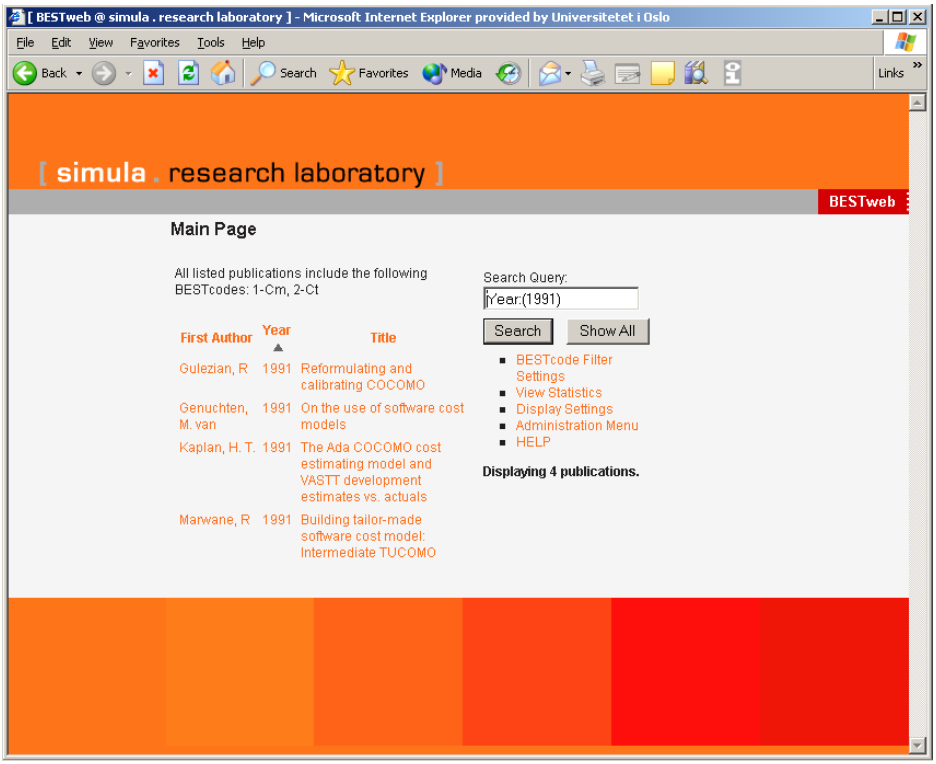


Figure 2 Screen shot of the main page in the BESTweb system

E.1.1 Startup

You log into the system the BESTweb system by click the "<<< Enter the BESTweb system." link on the initial page (default page: http://someUrl/BESTweb/); a log-in window will then appear requesting a username and password before access to the system is granted. Upon successful authentication the main page will be displayed. On this page you will find:

- A list of publications (by default, all publications in the BESTweb library are displayed).
- The number of publications being displayed.
- A text box where users type in their search query.
- The "Search" button for limiting the list of publications to those that contain the search query.
- The "Show All" button is used for listing all the publications in the BESTweb library (implicitly clearing the search query and BEST-codes filter settings).

- A link to filter (limit) the list of publications to those that contain specific BEST-codes (by default, none are chosen).
- A link to view statistics based on the current list of publications.
- A link to alter the display settings. For example, if the user is not interested in seeing the year of the publications, they can hide this information. The display settings allow the user to show/hide every piece of data associated with the publication.
- If the user has administrator privileges they will see the "Administration Menu" link. This menu allows an administrator to add/remove users and upload a new BESTweb library file.
- The "HELP" link brings the user to the Help Page which gives information on performing advanced searching.

E.1.2 The Publications List

The publications list shows the result of the intersection of the following three sets:

- All the publications from the BESTweb library.
- The publications that match the search query (if a search query is specified).
- The publications that contain at least one of the selected BEST-codes (from the BESTcodes filter settings page, if any are specified).

A message specifying that there are no publications to display is shown on the main page when:

1. A search results in no matching publications.
2. If the BESTweb library file contains no publications.
3. No publications in the library match the BEST-code filter settings.

E.1.1.1 Sorting

By clicking on any of the publication list headers at the top of the main page (e.g. First Author, Year, Title), the displayed publications are sorted in the ascending order with respect to the data in that header; an up-arrow appears besides the selected header name. If the same header is selected again the publications are sorted in the descending order, and a down-arrow appears besides the header.

E.1.3 The Publications

When a publication is selected a "Publication Details" page comes up showing all the data associated with that particular publication. If a selected publication has been classified with

the BEST codes then these codes will also be shown. A user selects a publication from the main page by clicking on any of the fields associated with a particular publication, for example, the title or the year.

E.1.4 Display Settings

By clicking the "Display Settings" link, the user is forwarded to the "Publications List Display Settings" page. Here a user can select the publication related information they wish to see on the main page (on the list of publications). The choices are not saved, they are discarded forgot when the session expires.

E.1.5 BEST-codes and BEST-code Filter Settings

Some of the papers (the software journal papers) in the library are coded according to the following categories: 1) research topic, 2) estimation approach, 3) research approach, 4) study context, and 5) data set. Each category has sub-categories. For example, a paper may be classified as 1-Em, 2-CtRg, 3-DmHv, 4-Pr, 5-BcKe. This means that the research topic is Em (Estimation method), the estimation approach is Ct (CART) and Rg (Regression), the research approach is Dm (development of a new method) and Hv (history-based evaluation), the study context is Pr (Professionals), and the applied data sets are Bc (Boehm) and Ke (Kemerer).

The term "BEST-code category" refers to, for example, "1) research topic". The term "BEST-code" refers to the combination of a category and a sub-category, for example the following are two BEST-codes, "2-Ct" and "2-Rg".

The BEST codes are used to ease the search for relevant papers. For example, when searching for all papers on the CART estimation method it is sufficient to search for all papers coded with Ct in the category estimation approach. This is accomplished by ticking the Ct field on the "BESTcodes Search Filter Settings" page. The publications list is filtered against the BEST-codes in an OR-manner, i.e., if more than one BEST code is ticked the search will find papers that have at least one of codes (as opposed to an AND-manner of filtering where the publications would have to have all the selected BEST-codes).

This "BESTcodes Search Filter Settings" page is accessed from the main page by clicking the "BESTcode Filter Settings" link. This page show all the BEST codes and categories being used by the BESTweb system. Once a user selects the BEST-codes they're interested in, he or she then selects the OK button. This brings the user back to the main screen which now shows the list of publications with the BEST-codes selection applied. The selected

BEST-code filter settings are displayed on the main page. The list of the selected BEST-codes is also shown on that page.

The “Show All” button automatically clears any selected BEST-codes.

E.1.6 Searching

A user performs a search by writing a search query in the text field on the main page and subsequently pressing the search button. One can use wildcards in the search, except as the first character in a search query. One can also confine the search to a specific field by writing: “header:(some query)”, e.g. Year:(1979), or Title:(interval). To search for a specific author use the Author keyword, e.g.: Author:(Abdel).

Note that when there are BEST-code filter settings chosen (see above for info on filter settings) the search results are an intersection of the BESTcode filter results and the search query results; i.e. a search will only return results that also satisfies the BESTcode filter settings.

The “Show All” button automatically clears any specified search query.

E.1.7 Viewing Statistics

By clicking the “View Statistics” link the user is brought to the “Statistics Options for the Current Query” page. Here there are two categories of statistics one can view:

1. Publication per year
2. Publications per a specific BEST-code category

The graphical statistics shown when clicking on any of these links are based on the publications that are currently being displayed on the main page.

E.1.8 Administration Menu

From the administration menu an administrator can add/remove BESTweb system users and upload a new BESTweb library file.

E.1.1.2 Adding a User

To add a user to the system the administrator (admin) selects the “Add a user to the BESTweb system” link. The admin is then forwarded to a screen where he or she specifies the username, full name, password and the type of the new account (normal or administrator); if a username is already in use an error message is displayed.

E.1.1.3 Removing a User

To remove (delete) a user from the system the admin selects the "Remove a user from the BESTweb system" link. The admin is then forwarded to a screen where all the usernames are listed. The admin must then click on the username they wish to remove. Upon deletion, the admin is forwarded back to the main page.

E.1.1.4 Upload a Library File

To upload a new BESTweb library file to the system the admin selects the "Upload a new database (XML) file" link. This brings the admin to the "BESTweb File Upload" page. On this page a BESTweb library file is specified using the "Browse" button. The file is then uploaded to the system using the "Submit" button.

Once the file is uploaded the user is forwarded to a page where results of the upload are shown. The page reports on the following:

- The number of successfully extracted publications.
- The number of publications that have some warnings (minor problems).
- The number of publications that have errors. These publications are skipped.
- (Optional) A list of publications that had warnings along with the warning(s).
- (Optional) A list of publications that had errors along with the error(s).

A publication is considered to have a major error (and will be skipped) if:

- The reference type is missing or unknown.
- The publication ID (EndNote reference number) is missing.
- The publication ID is used more than once.
- The publication has no authors.
- The year is not specified.
- The title is not specified.
- The secondary title is not be specified if a publication is of one of the following types: journal, conference, chapter, and magazine.

Warnings are displayed when an unsupported publication field is used. Also, a warning is shown if a publication is tagged (categorized / labeled) with an unknown BEST code or category.

The supported fields are: reference type, reference number (the publication ID), authors, year, title, place published, publisher, volume, number of volumes, number, pages, section, edition, date, type of work, ISBN, accession number, keywords (may only be used to specify the BEST codes), abstract, notes, and URL.

After a library file is uploaded, all logged-in users will see the following message the next time that they load the main page:
Please note: A new BESTweb library database has been uploaded to the system and the complete list of publications from this new library is now being shown. Click the "Search" button to apply the search query and the BEST-codes filter settings to this list.

E.1.9 Robustness

This section describes the robustness of the BESTweb system. A lot of these errors can be generated when the user manually modifies the URL.

E.1.1.5 At System Startup

If the BESTweb library file (specified in the bestweb.properties file via the LibraryFile key) is missing: access to the system is restricted to users with admin rights. Once an admin logs in he or she is forwarded to the administration menu so that she or he can upload the library file. Normal users are forwarded to a page that notifies them of the fact that the system is not ready for usage.
If the search index files are missing from the specified search index directory (specified in the bestweb.properties file via the IndexDir key) then the system will attempt to create the search index. The system will abort the startup if the IndexDir is unspecified or does not exist.
During system startup the following errors are considered fatal and will cause the system to abort the startup:

1. Problems accessing the database.
2. Problems accessing the search index or the search index directory.
3. Problems reading the BESTweb library file.
4. Problems reading bestweb.properties configuration file.

E.1.1.6 Viewing Publications Details

When a user requests to see the details of a specific publication the following errors are possible with respect to the publication ID (of the publication that they wish to see details on):

1. The ID is unknown, i.e. no publication is associated with the id. The user is notified that there is no publication associated with the specified publication ID.
2. The ID is missing, i.e. the request did not specify a publication ID. The user is redirected to the main page.

3. The ID is malformed, i.e. the specified ID is not of integer type. The user is redirected to the main page.

E.1.1.7 Sorting

If a request to sort the publications list does not specify a sorting order or an invalid sorting order is specified then the user is redirected to the main page.

E.1.1.8 Searching

If a search query is malformed, i.e. it cannot be parsed, an error messages is given.

E.1.1.9 Viewing BEST-code Category Statistics

The following errors will redirect the user to the main page:

1. The specified BEST-code category is unknown; i.e. the category does not exist in the system.
2. The BEST-code category is missing; i.e. no category is specified.
3. The BEST-code category is malformed; i.e. the specified category is not of integer type.

E.1.1.10 User Management

If during, the addition or removal of BESTweb users a database error is encountered, for example if a connection to the database cannot be established, an error message is shown.

E.1.1.11 Uploading of a BESTweb Library File

If the file BESTweb library file does not exist (incorrect filename), is zero-sized, or is not in the expect format, an error message is displayed to the admin.

E.2 Technical information

BESTweb workstation's configuration:

- Tomcat directory (this is the value of `$CATALINA_HOME`): C:\jakarta-tomcat-5.0.28
- MySQL directory: C:\mysql-4.0.21-win
- The BESTweb library file (BEST-library.xml) directory: C:\BESTweb
  - Note that this directory also contains BEST-library.xml.backup, a backup copy of the file.
- The search index directory: C:\BESTweb\index
- The log directory is: C:\BESTweb\logs
- Tomcat is running on port 8080.

The application is running at the following address: http://localhost:8080/BESTweb

E.3 Architecture

The system uses the Apache Struts 1.1 web application framework and is run in the Apache Tomcat 5 servlet container. Classes that depend on the Struts framework are located in the `no.simula.bestweb.web` package. The application uses the BESTweb library file (an XML file) as the "library" of the publications (see Section E.4 for the XSD: EndNote.xsd). This file is specified in the bestweb.properties file under the key `LibraryFile`, see Section E.1.13. The `LibraryFile` is initially uploaded to the system by a user selecting the "upload file" function from the administration menu. Once the file is uploaded the `no.simula.bestweb.parser` package in the system takes care of converting the content of the file into objects (instances of classes found in the `no.simula.bestweb.pubmdl` package). As the publications are extracted from the library file they are analyzed for critical content, i.e. content that is mandatory for the publication to not be considered erroneous, e.g. the title of the publication. One of the elements of a publication are the BEST codes, these are the classifications of the publication. These codes are optional, but if they are present they have to conform to the codes present in the BESTweb database. An object form of the codes in the database is built when the application first starts by functionality found in the `no.simula.bestweb.db` package. The database (`bestweb`) is powered by MySQL 4, it is used to store the BEST codes & categories, and the BESTweb system user information.

After an OO model of the file is built the searching index is constructed using the `no.simula.bestweb.index` package. This package relies on Jakarta Lucene, a high-performance, full-featured text search engine library written entirely in Java. After this step is complete the system is ready for usage. The search index directory is specified in the bestweb.properties file under the key `IndexDir`.

Note that the `LibraryFile` is read (an OO model of the file is built) on ever system starts-up, but the search index is only built when an administrator uploads a new file to system.

The application relies on Tomcat's basic authentication with MD5 encoding for the passwords.

The system also relies on the following third-party libraries:

- Cewolf: A chart enabling web object framework. Used to create graphical statistics of the user's search query results.
- JavaServer Pages Standard Tag Library (JSTL): Provides a library of powerful tags.

- Apache XMLBeans: An XML binding tool. Used to extract content from BESTweb-library.xml.
- Commons File Upload
- MySQL Connector
- Log4j
- Commons Codec: For MD5 password encoding.

The application's directory structure follows the style recommended by the Struts framework.

The `bestweb` database consists of the following tables:

- `users`: The list of BESTweb system users.
  - Columns: `user_name`, `full_name`, `user_pass`, `query`
- `roles`: The roles the users can have (i.e. admin and user).
  - Column: `rolename`
- `user_roles`: Specifies the roles each user is in.
  - Columns: `user_name`, `role_name`
- `code_categories`: The BEST-code categories.
  - Columns: `category_id`, `description`
- `codes`: The BEST-codes.
  - Columns: `id`, `code`, `category`, `description`

E.4 EndNote.xsd

The BESTweb-library.xml file corresponds to this XSD.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="XML" type="XmlType"/>
 <xsd:complexType name="XmlType">
  <xsd:sequence>
   <xsd:element name="RECORDS" type="RecordsType"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="RecordsType">
  <xsd:sequence>
   <xsd:element name="RECORD" type="RecordType" minOccurs="1"
maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="RecordType">
  <xsd:sequence>
   <xsd:element name="REFERENCE_TYPE" minOccurs="0">
    <xsd:simpleType>
     <xsd:restriction base="xsd:int">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="31"/>
```

```
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
   <xsd:element name="REFNUM" minOccurs="0">
    <xsd:simpleType>
     <xsd:restriction base="xsd:int">
      <xsd:minInclusive value="0"/>
     </xsd:restriction>
    </xsd:simpleType>
   </xsd:element>
   <xsd:element name="AUTHORS" minOccurs="0">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="AUTHOR" minOccurs="1" maxOccurs="unbounded"
type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="YEAR" type="xsd:string" minOccurs="0"/>
   <xsd:element name="TITLE" type="xsd:string" minOccurs="0"/>
   <xsd:element name="SECONDARY_AUTHORS" minOccurs="0">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="SECONDARY_AUTHOR" minOccurs="1"
maxOccurs="unbounded" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="SECONDARY_TITLE" type="xsd:string" minOccurs="0"/>
   <xsd:element name="PLACE_PUBLISHED" type="xsd:string" minOccurs="0"/>
   <xsd:element name="PUBLISHER" type="xsd:string" minOccurs="0"/>
   <xsd:element name="VOLUME" type="xsd:string" minOccurs="0"/>
   <xsd:element name="NUMBER_OF_VOLUMES" type="xsd:string"
minOccurs="0"/>
   <xsd:element name="NUMBER" type="xsd:string" minOccurs="0"/>
   <xsd:element name="PAGES" type="xsd:string" minOccurs="0"/>
   <xsd:element name="SECTION" type="xsd:string" minOccurs="0"/>
   <xsd:element name="TERTIARY_AUTHORS" minOccurs="0">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="TERTIARY_AUTHOR" minOccurs="1"
maxOccurs="unbounded" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="TERTIARY_TITLE" type="xsd:string" minOccurs="0"/>
   <xsd:element name="EDITION" type="xsd:string" minOccurs="0"/>
   <xsd:element name="DATE" type="xsd:string" minOccurs="0"/>
   <xsd:element name="TYPE_OF_WORK" type="xsd:string" minOccurs="0"/>
   <xsd:element name="SUBSIDIARY_AUTHORS" minOccurs="0">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="SUBSIDIARY_AUTHOR" minOccurs="1"
maxOccurs="unbounded" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="SHORT_TITLE" type="xsd:string" minOccurs="0"/>
   <xsd:element name="ALTERNATE_TITLE" type="xsd:string" minOccurs="0"/>
   <xsd:element name="ISBN" type="xsd:string" minOccurs="0"/>
   <xsd:element name="ORIGINAL_PUB" type="xsd:string" minOccurs="0"/>
```

```
    <xsd:element name="REPRINT_EDITION" type="xsd:string" minOccurs="0"/>
    <xsd:element name="REVIEWED ITEM" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CUSTOM1" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CUSTOM2" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CUSTOM3" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CUSTOM4" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CUSTOM5" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CUSTOM6" type="xsd:string" minOccurs="0"/>
    <xsd:element name="ACCESSION NUMBER" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CALL_NUMBER" type="xsd:string" minOccurs="0"/>
    <xsd:element name="LABEL" type="xsd:string" minOccurs="0"/>
    <xsd:element name="KEYWORDS" minOccurs="0">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="KEYWORD" minOccurs="1" maxOccurs="unbounded"
type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="ABSTRACT" type="xsd:string" minOccurs="0"/>
    <xsd:element name="NOTES" type="xsd:string" minOccurs="0"/>
    <xsd:element name="URL" type="xsd:string" minOccurs="0"/>
    <xsd:element name="AUTHOR_ADDRESS" type="xsd:string" minOccurs="0"/>
    <xsd:element name="IMAGE" type="xsd:string" minOccurs="0"/>
    <xsd:element name="CAPTION" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

E.5 Configuration

This section describes how to configure Tomcat, MySQL, and system specific settings.

E.1.10 $CATALINA_HOME/conf/server.xml

Add the following context to $CATALINA_HOME/conf/server.xml:

- Place under:

```
    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
       debug="0" resourceName="UserDatabase"/>
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
  digest="MD5" driverName="com.mysql.jdbc.Driver"
  connectionURL="jdbc:mysql://localhost:3306/bestweb"
  connectionName="bestwebuser" connectionPassword="socamo03"
  userTable="users" userNameCol="user_name" userCredCol="user_pass"
  userRoleTable="user_roles" roleNameCol="role_name"/>
```

- Place under:

```
    <Logger className="org.apache.catalina.logger.FileLogger"
       directory="logs"  prefix="localhost_log." suffix=".txt"
       Time-Stamp="true"/>
<Context path="/BESTweb" reloadable="true">
  <Resource name="jdbc/UserDB" reloadable="true"
            auth="Container" type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/bestwebDB">
    <!-- Preventing dB connection pool leaks -->
    <parameter>
```

```
      <name>removeAbandoned</name>
      <value>true</value>
    </parameter>
    <parameter>
      <name>removeAbandonedTimeout</name>
      <value>60</value>
    </parameter>
    <parameter>
      <name>logAbandoned</name>
      <value>true</value>
    </parameter>

    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>
    <!-- Maximum number of dB connections in pool. Make sure you
         configure your mysqld max_connections large enough to handle
         all of your db connections. Set to 0 for no limit.
    -->
    <parameter>
      <name>maxActive</name>
      <value>100</value>
    </parameter>
    <!-- Maximum number of idle dB connections to retain in pool.
         Set to 0 for no limit.
    -->
    <parameter>
      <name>maxIdle</name>
      <value>10</value>
    </parameter>
    <!-- Maximum time to wait for a dB connection to become available
         in ms, ex: 10000 = 10 seconds. An Exception is thrown if
         this timeout is exceeded.  Set to -1 to wait indefinitely.
     -->
    <parameter>
      <name>maxWait</name>
      <value>10000</value>
    </parameter>
    <!-- MySQL dB username and password for dB connections  -->
    <parameter>
      <name>username</name>
      <value>bestwebuser</value>
    </parameter>
    <parameter>
      <name>password</name>
      <value>socamo03</value>
    </parameter>
    <parameter>
     <name>driverClassName</name>
     <value>com.mysql.jdbc.Driver</value>
    </parameter>
    <parameter>
      <name>url</name>
      <value>
        jdbc:mysql://localhost:3306/bestweb?autoReconnect=true
      </value>
    </parameter>
  </ResourceParams>

</Context>
```

E.1.11 $CATALINA_HOME/common/lib

The application relies on the following JARs being present in `$CATALINA_HOME/common/lib`:

- The MySQL ConnectorJ JAR, e.g. `mysql-connector-java-3.0.11-stable-bin.jar`.

E.1.12 The Database Configuration

To configure the database follow the following steps:

1. C:\mysql-4.0.21-win\bin> mysql -u root
2. create database bestweb;
3. grant all privileges on bestweb.* to bestwebuser@localhost identified by 'socamo03' with grant option;
4. exit
5. mysql -u root bestweb < bestweb.sql

Note, the bestweb.sql is located in the C:\mysql-4.0.21-win\bin directory.

To reset the database invoke the reset_bestweb.bat file, also located in C:\mysql-4.0.21-win\bin.

E.1.13 bestweb.properties

The `bestweb.properties` file, located in the `WEB-INF/classes/resources` directory specifies two important settings:

- The BESTweb library file.
- The directory that will be used to hold the searching index.

The following is an example of the contents of the file:

```
LibraryFile=C:/BESTweb/BEST-library.xml
IndexDir=C:/BESTweb/index
```

Bibliography

[1] K. Beck, *Extreme Programming Explained*: Addison-Wesley, 2000.

[2] R. S. Pressman, *Software Engineering - A Practitioner's Approach*, 7th ed.: McGraw Hill, 2005.

[3] R. Glass, *Facts and Fallacies of Software Engineering*: Addison-Wesley Professional, 2002.

[4] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture--Practice and Promise*: Addison-Wesley Professional, 2003.

[5] T. McGibbon, "Software Reliability Data Summary," Data Analysis Center for Software 1996.

[6] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, "What We Have Learned About Fighting Defects," in *Eighth IEEE International Symposium on Software Metrics (METRICS'02)*, 2002.

[7] W. J. Dzidek, E. Arisholm, and L. Briand, "A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance," *IEEE Transactions on Software Engineering,* vol. 34, pp. 407-432, May 2008.

[8] ISO/IEC, "ISO/IEC 12207:1995, Information Technology—Software Life-cycle Processes," 1995.

[9] V. T. Rajlich and K. H. Bennett, "A Staged Model for the Software Life Cycle," *IEEE Computer,* vol. 33, pp. 66-71, 2000.

[10] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering,* vol. 27, pp. 1-12, 2001.

[11] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*: Addison-Wesley, 1999.

[12] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 2nd ed.: Prentice Hall, 2004.

[13] E. Dijkstra, "Go To Statement Considered Harmful," *Communications of the ACM,* vol. 11, pp. 147-148, 1968.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley Professional, 1995.

[15] J. Eder, G. Kappel, and M. Schrefl, "Coupling and Cohesion in Object-Oriented Systems," in *Conference on Information and Knowledge Management*, Baltimore, USA, 1992.

[16] B. Meyer, *Object-Oriented Software Construction* vol. 2nd: Prentice Hall, 2000.

[17] E. Arisholm and D. I. K. Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Transactions on Software Engineering,* vol. 30, pp. 521-534, 2004.

[18] OMG, "Unified Modeling Language Specification 1.4," 2001, http://www.omg.org/spec/UML/1.4/.

[19] OMG, "UML 2.0 Infrastructure and Superstructure," 2005, http://www.omg.org/spec/UML/2.0/.

[20] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed.: Addison-Wesley, 1994.

[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorenzen, *Object-Oriented Modeling and Design*: Prentice-Hall, 1991.

[22] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*: Addison-Wesley, 1992.

[23] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA* 2nd ed.: Addison Wesley, 2003.
[24] M. Fowler and K. Scott, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd ed. Reading, Mass.: Addison-Wesley, 2000.
[25] P. Baker, S. Loh, and F. Weil, "Model-Driven Engineering in a Large Industrial Context — Motorola Case Study," in *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005*, 2005.
[26] S. Cook, "Domain-Specific Modeling and Model Driven Architecture," 2004, http://www.bptrends.com/publicationfiles/01-04 COL Dom Spec Modeling Frankel-Cook.pdf.
[27] D. Thomas, "MDA: Revenge of the Modelers or UML Utopia?," *IEEE Software,* vol. 21, 2004.
[28] W. J. Dzidek, L. C. Briand, and Y. Labiche, " Lessons Learned from Developing a Dynamic OCL Constraint Enforcement Tool for Java," in *Satellite Events at the MoDELS 2005 Conference*. vol. 3844, J.-M. Bruel, Ed.: Springer-Verlag GmbH, 2006, pp. 10-19.
[29] L. C. Briand, W. J. Dzidek, and Y. Labiche, "Instrumenting Contracts with Aspect-Oriented Programming to Increase Observability and Support Debugging," in *21st IEEE International Conference on Software Maintenance (ICSM)*, Budapest, Hungary, 2005, pp. 687-690.
[30] P. Baker, Z. R. Dai, J. Grabowski, Ø. Haugen, I. Schieferdecker, and C. Williams, *Model-Driven Testing: Using the UML Testing Profile*: Springer, 2007.
[31] S. Ali, L. C. Briand, M. J.-u. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, "A state-based approach to integration testing based on UML models," *Information and Software Technology,* vol. 49, pp. 1087-1106, 2007.
[32] L. C. Briand, Y. Labiche, and J. Leduc, "Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software," *IEEE Transactions on Software Engineering,* vol. 32, pp. 642-663, September 2006.
[33] W. A. Babich, *Software Configuration Management*: Addison Wesley Longman, 1986.
[34] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*: Kluwer, 2000.
[35] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen, " The Future of Empirical Methods in Software Engineering Research," in *Future of Software Engineering, 2007. FOSE '07*, 2007, pp. 358-378.
[36] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," vol. 12, pp. 733-743, 1986.
[37] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," in *IEEE Computer*. vol. 31, 1998, pp. 23-31.
[38] A. Pinsonneault and K. L. Kraemer, "Survey Research Methodology in Management Information Systems: An Assessment," *Journal of Management Information Systems,* vol. 10, pp. 75–105, 1993.
[39] J. Segal, "When Software Engineers met Research Scientists: A Case Study," *Empirical Software Engineering,* vol. 10, pp. 517-536, 2005.
[40] R. K. Yin, *Case Study Research Design and Methods*, 2nd ed.: Sage, 1994.
[41] V. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering,* vol. 25, pp. 456-473, 1999.

[42] V. Basili, "The Role of Experimentation in Software Engineering: Past, Current, and Future," in *IEEE International Conference on Software Engineering*, 1996, pp. 442-449.
[43] D. Blake, "Programming Languages: Everyone Has a Favorite One," in *Dr. Dobb's*, 2008.
[44] M. Höst, B. Regnell, and C. Wohlin, "Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Engineering,* vol. 5, pp. 201-214, 2000.
[45] P. Berander, "Using Students as Subjects in Requirements Prioritization," in *International Symposium on Empirical Software Engineering (ISESE'04)*, 2004, pp. 167-176.
[46] E. B. Alex, "Death by UML Fever," vol. 2, pp. 72-80, 2004.
[47] B. Kitchenham, "Procedures for undertaking systematic reviews," Department of Computer Science, Keele University and National ICT, Australia Ltd 2004.
[48] J. Bloch, *Effective Java Programming Language Guide*: Prentice Hall PTR, 2001.
[49] M. Jørgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Transactions on Software Engineering,* vol. 33, pp. 33--53, January 2007.
[50] Thomson-Scientific, "Web of Science," 2007.
[51] O. R. Holsti, *Content analysis for the social sciences and humanities*. Reading, Mass.: Addison-Wesley Pub. Co., 1969.
[52] M. Grossmana, J. E. Aronsonb, and R. V. McCarthy, "Does UML make the grade? Insights from the software development community," *Information and Software Technology,* vol. 46, pp. 383-397, 2005.
[53] C. F. J. Lange and M. R. V. Chaudron, "Effects of defects in UML models: an experimental investigation," in *International Conference on Software Engineering*, 2006.
[54] B. Anda and K. Hansen, "A case study on the application of UML in legacy development," in *Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering* Rio de Janeiro, Brazil: ACM, 2003.
[55] M. Tsagias and B. Kitchenham, "An evaluation of the business object approach to software development " *The Journal of Systems and Software,* vol. 52, pp. 149-156 2000.
[56] J. S. Lim and S. R. Jeong, "An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software," *The Journal of Systems and Software,* vol. 77, pp. 131-138, 2005.
[57] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation," *IEEE Transactions on Software Engineering,* vol. 32, pp. 365-381, 2006.
[58] S. Ceria and J. J. Cukier, "Use Case Driven Iterative Development: Hurdles and Solutions," in *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006*, 2006.
[59] M. Torchiano, F. Ricca, and P. Tonella, "A comparative study on the re-documentation of existing software: Code annotations vs. drawing editors," in *International Symposium on Empirical Software Engineering*, 2005.
[60] C. F. J. Lange, B. D. Bois, M. R. V. Chaudron, and S. Demeyer, "An Experimental Investigation of UML Modeling Conventions," in *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, 2006.

[61] L. C. Briand, Y. Labiche, M. D. Penta, and H. D. Yan-Bondoc, "An Experimental Investigation of Formality in UML-Based Development," *Transactions on Software Engineering,* vol. 31, pp. 833-849, 2005.
[62] M. C. Otero and J. J. Dolado, "An Initial Experimental Assessment of the Dynamic Modelling in UML," *Empirical Software Engineering,* vol. 7, pp. 27 - 47, 2002.
[63] M. C. Otero and J. J. Dolado, "Evaluation of the comprehension of the dynamic modeling in UML," *Information and Software Technology,* vol. 46, pp. 35-53, 2004.
[64] J. Swan, T. Barker, C. Britton, and M. Kutar, "An empirical study of factors that affect user performance when using UML interaction diagrams," in *International Symposium on Empirical Software Engineering, 2005*, 2005.
[65] C. Glezer, M. Last, E. Nachmany, and P. Shoval, "Quality and comprehension of UML interaction diagrams-an experimental comparison," *Information and Software Technology,* vol. 47, pp. 675–692, 2005.
[66] J. A. Cruz-Lemus, M. Genero, M. E. Manso, and M. Piattini, "Evaluating the Effect of Composite States on the Understandability of UML Statechart Diagrams," in *ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems*, 2005.
[67] C. Bunse, "Using patterns for the refinement and translation of UML models: A controlled experiment," *Empirical Software Engineering,* vol. 11, pp. 227 - 267, 2006.
[68] L. Kuzniarz, M. Staron, and C. Wohlin, "Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments," *Journal of Systems and Software,* vol. 79, pp. 727-742, 2006.
[69] L. Reynoso, M. Genero, M. Piattini, and E. Manso, "Assessing the Impact of Coupling on the Understandability and Modifiability of OCL Expressions within UML/OCL Combined Models," in *11th IEEE International Software Metrics Symposium (METRICS 2005)*, 2005.
[70] B. Anda, H. Dreiem, D. I. K. Sjøberg, and M. Jørgensen, "Estimating Software Development Effort Based on Use Cases – Experiences from Industry," in *4th International Conference on the Unified Modeling Language (UML2001)*, 2001.
[71] M. Genero, M. Piatini, and E. Manso, "Finding "Early" Indicators of UML Class Diagrams Understandability and Modifiability," in *Proceedings of the 2004 International Symposium on Empirical Software Engineering*, 2004.
[72] M. G. Bocco, D. L. Moody, and M. Piattini, "Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 17, pp. 225 - 246, 2005.
[73] M. Genero, M. Piattini, E. Manso, and G. Cantone, "Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics," in *Proceedings of the 9th International Symposium on Software Metrics*, 2003.
[74] C. F. J. Lange, M. R. V. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," in *IEEE Software*. vol. 23, 2006, pp. 40-46.
[75] F. S. Roberts, *Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences*: Encyclopedia of Mathematics and its Applications Addison Wesley Publishing Company, 1979.
[76] H. Zuse, *History of Software Measurement by Horst Zuse*, 1995.
[77] E. Rudolph, P. Graubmann, and J. Grabowski, "Tutorial on Message Sequence Charts," *Computer Networks and ISDN Systems,* vol. 28, pp. 1629-1641, 1996.
[78] J. Wüst, "The software design metrics tool for the UML," 1.3 ed.

[79] Telelogic, "TAU," 2003.
[80] Microsoft, "Visio," 2002.
[81] Omondo, "Omondo," 2005.
[82] P. Kruchten, *The Rational Unified Process: An Introduction*: Addison-Wesley, 2003.
[83] B. Selic, "The Pragmatics of Model-Driven Development," *IEEE Software,* vol. 20, pp. 19-25, September/October 2003.
[84] OMG, "UML 2.0 Diagram Interchange Specification," 2004.
[85] Borland, "Borland Together for Eclipse," 2004.
[86] J. Holmes, *Struts: the complete reference*. New York; London: McGraw-Hill, 2004.
[87] Sun-Microsystems, "JavaServer Pages 2.0 Specification," 2003.
[88] J. Gosling, *The Java language specification*, 2nd ed. Boston: Addison-Wesley, 2000.
[89] T. A. Powell, *HTML : the complete reference*, 3rd ed. Berkeley, Calif.: Osborne/McGraw-Hill, 2001.
[90] IBM, "Eclipse," 2004.
[91] M. Kofler, *MySQL*. Berkeley, CA: Apress, 2001.
[92] Simula-Research-Laboratory, "BESTweb - http://simula.no/BESTweb/," 2004.
[93] R. Binder, *Testing object-oriented systems : models, patterns, and tools*. Reading, MA: Addison-Wesley, 2000.
[94] J. Tian, *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*: John Wiley and Sons, Inc., and IEEE Computer Society Press, 2005.
[95] J. L. Devore and N. Farnum, *Applied Statistics for Engineers and Scientists*: Duxbury, 1999.
[96] L. V. Garcia, "Escaping the Bonferroni iron claw in ecological studies," *Oikos,* vol. 105, pp. 657-663, 2004.
[97] J. Cohen, *Statistical power analysis for the behavioral sciences*, 2nd ed. Hillsdale, NJ: L. Erlbaum Associates, 1988.
[98] R. H. Myers, D. C. Montgomery, and G. G. Vining, *Generalized linear models : with applications in engineering and the sciences*. New York: J. Wiley, 2002.
[99] S. E. Hove and B. Anda, "Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research," in *11th IEEE International Software Metrics Symposium (METRICS'05)* Como, Italy: IEEE, 2005.
[100] S. Meyers, *Effective C++*, 2nd ed.: Addison-Wesley Professional, 1997.
[101] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review,* vol. 63, pp. 81-97, 1956.
[102] W. R. Shadish, Cook Thomas D., and Campbell Donald Thomas, *Experimental and quasi-experimental designs for generalized causal inference*. Boston [u.a.]: Houghton Mifflin, 2002.
[103] S. Tilley and S. Huang, "A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding," in *21st Annual International Conference on Systems Documentation*, 2003, pp. 184-191.
[104] E. Tryggeseth, "Report from an Experiment: Impact of Documentation on Maintenance," *Empirical Software Engineering Journal,* vol. 2, pp. 201-207, 1997.
[105] T. H. Huxley, "We are all scientists," in *The New Treasury of Science*, H. Shapley, S. Rapport, and H. Wright, Eds. London and Glasgow: Collins, 1965.

[106] A. Brooks, M. Roper, M. Wood, J. Daly, and J. Miller, "Replication's Role in Software Engineering," in *Guide to Advanced Empirical Software Engineering*, F. Schull, J. Singer, and D. Sjoberg, Eds.: Springer-Science, 2008, pp. 365-379.
[107] B. Curtis, "Measurement and experimentation in software engineering," *Proceedings of the IEEE,* vol. 68, pp. 1144–1157, 1980.
[108] K. R. Popper, *The Logic of Scientific Discovery*. London: Hutchinson, 1968.
[109] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal, "A Survey of Controlled Experiments in Software Engineering," *IEEE Transactions on Software Engineering,* vol. 31, pp. 1-21, 2005.
[110] H. M. Collins, *Changing Order Replication and Induction in Scientific Practice*. London: SAGE Publications, 1985.
[111] W. Broad and N. Wade, *Betrayers of the Truth*. New York: Oxford University Press, 1986.
[112] J. Brewer and A. Hunter, *Multimethod Research: A Synthesis of Styles*. Newbury Park, CA: SAGE Publications, 1989.
[113] E. Arisholm, H. E. Gallis, T. Dybå, and D. I. K. Sjøberg, "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise," *IEEE Transactions on Software Engineering,* vol. 33, pp. 65-86, 2007.
[114] IBM, "Rational Software Architect," 2007.
[115] SAS, "SAS 9.1," 2004.