An Initial Controlled Experiment to Evaluate the Effect of UML Design Documentation on the Maintainability of Object Oriented Software in a Realistic Programming Environment

Cand. scient. thesis

Samera Afsheen Ali

12th January 2004
Abstract
The Unified Modeling Language (UML) is becoming the standard notation for expressing object-oriented analysis and design models. However, to date, there are only very few empirical studies that have attempted to evaluate UML, that is, to evaluate the potential costs, benefits, and appropriate use of UML. For example, a common claim is that the UML documentation will improve the ease of understanding how to perform changes to the software it represents. This might be very important as the cost of maintaining software is very high. However, there are only a few studies attempting to evaluate the actual benefits of UML during program comprehension. Most of these existing studies have been based on subjective assessments of the value of UML as opposed to more objective indicators of the benefits such as development time savings or improved quality during actual software development in a controlled setting. Furthermore, using UML may incur additional costs, because the UML documents need to be developed and maintained. Thus, the tradeoffs between the costs and benefits of using UML need to be investigated.

This thesis describes a controlled experiment to evaluate the effect of UML documentation on the maintainability of object-oriented software. In the experiment, 20 students designed and coded several Java maintenance tasks. About half of them received UML documentation, whereas the other half did not. The subjects who received UML documentation also had to update the documentation using a tool, Tau UML.

The primary goal of the experiment was to evaluate to which extent the access to UML class and sequence diagrams improves the ease of understanding and changing object-oriented software. To assess the costs and benefits of using UML in object-oriented development projects, the experiment also attempted to assess the additional costs associated with updating the UML documentation. The dependent variables of the study were effort (in minutes spent to solve the tasks) and correctness. The effort data was reported by each subject using a task questionnaire. The correctness of the task solutions was assessed by the researchers. The subjects spent between five and eight hours solving tasks. Five observers were present during the experiment.

The results of the experiment reported in this thesis indicate that having UML documentation may increase the chance of producing correct solutions by as much as 50 percent for the most
complex tasks, that is, for the tasks that require an in-depth understanding of the system to perform the changes to the code. For simpler maintenance tasks, the effect of having access to UML on program correctness is much smaller but still positive. The subjects who received UML documentation also spent less time on coding. However, because they also had to update the UML documents, the total effort to perform the changes was about 20 percent higher than for the subjects who did not receive UML documents.

In summary, the results of this thesis suggest that UML may have a positive effect on the maintainability of object-oriented software. The suggested 20 percent increase in change effort seem to be an acceptable amount of overhead as the benefits (savings) associated with less faulty software probably more than outweigh the additional costs incurred by updating the UML documentation. However, further research through replications is required to assess the extent to which the results can be generalized outside the experimental conditions, e.g., to professional developers solving real maintenance tasks: In this experiment, the subjects were students. Furthermore, the tasks and systems were small and perhaps not representative of “typical” maintenance tasks.
Acknowledgement

I like to express gratitude to my supervisor, Dr Erik Arisholm and Simula Research Laboratory for giving me an opportunity to conduct an experiment. Thanks to my supervisor for great motivation and guidance.

Secondly I would like to thank everyone involved in the experiment: Thanks to Gunnar Carelius, for all his help with SESE. Thanks to Eskild Busch, for helping out with Tau UML problems. Thanks to Siw Hove, for help with the logistic surrounding the experiment, and for conducting and analyzing the interviews. Many thanks to the students that participated in the Experiment.

Last, I want to thank my fellow students at the Department of Informatics at the University of Oslo, my family and friends. You have all been a great support. Thank You!

Samera Afsheen Ali
14.01.2003
Chapter 1 - Introduction

1.1 The Costs of Maintaining Object-Oriented Software

Software development projects consist of established steps, such as requirement analysis, design, programming, testing, implementing and maintaining. Among these steps the maintenance phase are the most costly (Table 1). After the delivery the development continues, making it better and faster, and more flexible (Table 2). The relative maintenance effort is between 50%-75% of the total effort.[10]. As indicated by Table 1, to correct a fault after delivery costs about 200 times more than if the fault was discovered during the initial requirements analysis.

<table>
<thead>
<tr>
<th>Relative cost of fixing mistakes</th>
<th>Continuous development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements x1</td>
<td>Perfective 63,3%</td>
</tr>
<tr>
<td>Design x5</td>
<td>(New functionality)</td>
</tr>
<tr>
<td>Coding x10</td>
<td>Adaptive 18,2%</td>
</tr>
<tr>
<td>Testing x20</td>
<td>(new platforms or OS)</td>
</tr>
<tr>
<td>Delivery x200</td>
<td>Corrective 17.4%</td>
</tr>
<tr>
<td></td>
<td>(fixing reported errors )</td>
</tr>
<tr>
<td></td>
<td>Other 4.1%</td>
</tr>
</tbody>
</table>

Table 1 Relative cost of fixing mistakes [10]

Table 2 Distribution of changes during maintenance [10]

1.2 Using UML to Reduce Software Maintenance Cost

So how can we reduce the maintenance costs? When the maintainer has the requirements and system structure fresh in mind, the addition of new functions can often be done based on the code. However, making changes to a system that the developers know is easier than making changes to systems that are completely unknown. Knowledge about a system is gained through system understanding. Previous research has identified two main categories that require understanding [11]:

- Continuous development
- Perfective (New functionality)
- Adaptive (new platforms or OS)
- Corrective (fixing reported errors)
- Other 4.1%
To obtain such understanding, documentation is needed. Anything written under the development process is essentially documentation: the source code, notes on the back of a sheet of paper, analysis reports, the design diagrams. There can also be a textbook (user guide) describing the different functions and their use. The essential software documentation includes [11]:

- Source code: includes programs, formal data models and other information that is needed to execute an operational system
- Design documentation: can be a text and/or diagrams, and it can be maintained manually or through tool support.
- Quality data: included test plans, data and results
- Management data: the plans and decision making reports

Clearly, the code is the most precise documentation, but it may be incomprehensible for large systems. To better gain an overview and fast understanding of the system for a newcomer, the design documentation might therefore be the most logical place to start. There are two ways to express the design documentation: you can either write text and explain the overall data structure textually, or you can use design diagrams expressed in a design language or notation such as the Unified Modeling Language (UML) [12].

UML is becoming the standard design notation for expressing object-oriented analysis and design models. It is conceivable that UML can be utilized to reduce the costs of maintaining object-oriented software. However, there are only very few empirical studies that have attempted to evaluate UML, studies that evaluate the potential costs, benefits and appropriate use of UML. For example, a common claim is that the UML documentation will improve the ease of understanding how to perform changes to the software it represents. However, using UML may incur additional costs, because the UML documents need to be developed and maintained. Thus, the tradeoffs between the potential costs and benefits of using UML need to be investigated.
1.3 The Experiment

This thesis describes a controlled experiment to evaluate the effect of UML documentation (versus no UML documentation) on the maintainability of object-oriented software. In this experiment, 20 students designed and coded several Java change (maintenance) tasks. About half of them received UML documentation, whereas the other half did not. The subjects who received UML documentation also had to update the documentation using a tool, Tau UML [7].

Goals

The primary goal of the experiment was to evaluate to which extent the access to UML class and sequence diagrams improves the ease of understanding and changing object-oriented software. To assess the costs and benefits of using UML in object-oriented development projects, the experiment also attempted to assess the additional costs associated with updating the UML documentation.

A secondary goal of the experiment was to evaluate how a so-called “think-aloud screen” affected the performance of subjects in controlled software engineering experiments, that is, whether the think-aloud screen decreases performance of the subjects (task solving time and correctness of the task solutions) compared with the silent condition, and whether the think-aloud screen provides valuable information that cannot be collected in the silent condition. This paper focuses on the evaluation of the effect of using UML to understand and change object-oriented software. Further discussion of the effects of using the think-aloud screen is discussed in [5].

Preparation

The experiment was built on the same design as in Arisholm et al.’s paper evaluating changeability on different design principles [2]. The source code used was the same, and so were the change tasks. The addition made for the new experiment was the design documentation. The source code was the java code for a coffee machine. The subject was to change this code to fit the customer requirements (the change tasks). The design was retrieved manually by analyzing the source code, and was then translated into UML class diagram and sequence diagrams. The diagrams were made in Tau UML [7], and tool used in teaching on
the University of Oslo. Individual Tau UML projects were made in for the students that were to receive UML documentation.

The experiment was performed using an online environment called SESE [3]. The questionnaires to the experiment were made available through this environment on the web. Four different versions of the experimental treatments were prepared. For example, those with UML had to get questions regarding their use of the UML design. Those with Think Aloud had to get up the Think Aloud screen every 20 minutes with the text ‘How have you been thinking while solving this task?’ , and so on.

**Execution**

There subjects were divided into four groups corresponding to different experimental treatments they were to receive:

1. Java code, no UML
2. Java code, UML
3. Java code, think aloud, no UML
4. Java code, think aloud, UML.

Except for these differences, all subjects performed the same coding tasks. Using SESE, the subjects downloaded and uploaded files through the web, they got a ZIP file containing their treatment. Those subjects that had UML had to be given access to Tau UML. In Tau UML they were given their own project, so that they did changes to there design documentation. The subjects spent between five and eight hours solving tasks. Five observers were present during the experiment.

**Analysis**

The time and the quality were the basic criteria for performing the analysis, that is, to determine the costs and benefits of using UML documentation during maintenance. The dependent variables of the study were **effort** (in minutes spent to solve the tasks) and **correctness**. The effort data was reported by each subject using a task questionnaire. The correctness of the task solutions was assessed by the researchers. In addition, interviews with all subjects were conducted, in an attempt to explain and triangulate the quantitative results.

After the experiment, the results had to be analyzed. SESE had stored the data in an MS Access database containing the times of the subjects and the answers to the questionnaires.
The source code solutions were stored in zip files by the subjects, before uploading their solutions to SESE. The source codes were analysed manually, and given 0 for wrong and 1 for right solutions.

1.4 Thesis Contribution

The contributions of this thesis are as follows.

The experiment provides initial knowledge regarding the costs and benefits of UML in a development context in which the developers have no prior knowledge of the systems to be maintained. Although further studies are required, such knowledge may eventually support decision making as to whether software development organizations should adopt the UML for their development projects.

The author of this thesis contributed to all aspects of this experiment, including establishing goals, planning, preparation and quality assurance of the experimental materials (including running a pilot experiment), recruiting subjects, experiment execution, analysis and presentation of results. A technical report of the experiment has been written by Erik Arishlom, Samera Afsheen Ali and Siw E. Hove. The report is the basis of this thesis.

Additionally, this thesis presents existing literature describing other relevant sources of information regarding the costs and benefits of using UML in development and as a means for program comprehension. The study is presented in Chapter 3.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 describes the UML and gives an overview of models and their uses, going in more detail on class and sequence diagrams. Chapter 3 shows the potential use of the UML, exemplified by the Rational Unified Process (RUP) and the use of UML during maintenance. Chapter 4 describes methods in empirical studies, focusing on controlled experiments. Chapter 5 is based on a technical report and describes the details of the experiment. Chapter 6 concludes.
Chapter 2 - UML Overview

2.1 Why UML

UML is a graphical language for visualizing, specifying, constructing and documenting an object-oriented software system. This language is said to ease the way to understand object-oriented software. UML specifies the structure and the system behavior. With UML, the developers can use the diagrams to help them understand the complexity of the software. A UML diagram helps to visualize the system during the planning and design phase of the development, to see if it meets the customer requirements. This approach is used in Rational Unified Process (RUP) [14]. UML can be used to communicate between the different stakeholders in the software development project (like customer and developer, system architect and database developer, and other developer that works on different stages of the project). UML has given everyone involved in a software development project a common vocabulary to communicate software design. A UML diagram is an abstraction of the problem and with a set of well defined diagrams, with precise notation, UML gives the opportunity to understand every stage of the project, at any given time of the project. The UML is a way for the original developer to communicate his design decisions to those who will take over and maintain the system.

2.2 UML Diagrams

There are two types of diagrams in UML, the static diagrams and the dynamic diagrams. The static diagram covers the software structure, while the dynamic diagrams show the interrelationship and behavior between the objects. There are a total of nine diagrams in UML. These are listed below.

The static diagrams:

- **Class Diagrams** – models the static data structure, showing the classes and the relationships between them. The class diagram gives an overview of which class has what attributes and which methods.
- **Object Diagram** – is an instance of a class diagram, showing a snapshot of the detailed state of a system at a given point in time. The object diagrams is not much used, it is mostly used to show examples of the data structures.
• **Component diagrams** – models the implementation and deployment of the system, and describes the dependencies between various software components

• **Deployment diagrams** – models the hardware used in implementing the system and the association between those hardware components. A component in a deployment diagram shows the location of their deployment.

**The dynamic diagrams:**

• **Sequence diagram** – describes the dynamic behavior of a set of objects arranged in time sequence. Sequence diagrams are good for real time specification and complex scenarios.

• **Use Case diagrams** – shows actors and use cases, and their relationships. The use cases represent functionality of a system. Use cases displays the different scenarios that accrue in the system

• **Collaboration diagram** – shows a cross between an object diagram and a sequence diagram. But does not model in time sequence.

• **Statechart diagram** – a state machine which describes the response of an object of a given class to the receipt of outside events

• **Activity diagrams** are used to document workflows in a system, from the business level down to the operational level. The Activity diagram is a variation of the state diagram where the "states" represent operations, and the transitions represent the activities that happen when the operation is complete. The general purpose of Activity diagrams is to focus on flows driven by internal processing vs. external events.

Among these diagrams the class diagram and the sequence diagram are described in more detail. Class diagram were used to **model** the static data of the system used in the experiment, sequence diagrams described the dynamic data. The diagrams that are used as examples in the following chapter are taken from the design documentation that was given the subjects in the experiment. The source code is for a coffee machine and a “minibank” (ATM). The rest of the diagrams can be viewed in appendix F.

### 2.3 Class Diagrams

Class diagrams are basically used to visualize, specify and document the static structure of the software system. A class diagram gives an overview of the system displaying the classes and the relationship among them. They show how the class interacts but not what happens when they interact.
Evaluating the Effect of UML on Maintainability

Notation:
The Class is a square box, divided in three, as shown in Figure 1. The top shows the class name, in the middle the attributes are listed, and on the bottom the methods are listed.

Some of the relationship between classes is summarized in Table 3:

- **Association**: An Association is a relationship between two classes, and is modeled by a line connecting the two classes. This line can be qualified with the type of relationship.

- **Composition**: If a class cannot exist by itself, and instead must be a member of another class, then that class has a composition relationship with the containing class. A Composition relationship is indicated by a line with a filled diamond.

- **Dependency**: When a class uses another class, perhaps as a member variable or a parameter, and so "depends" on that class, a Dependency relationship is formed. A Dependency relationship is indicated by a dotted arrow.

- **Aggregation**: Aggregations indicate a whole-part relationship, and are known as "has-a" relationships. An Aggregation relationship is indicated by a line with a hollow diamond.

- **Generalization**: A Generalization relationship is the equivalent of an inheritance relationship in object-oriented terms (an "is-a" relationship). A Generalization relationship is indicated by an arrow with a hollow arrowhead pointing to the base, or "parent", class.

Table 3: Relationships between Classes in a Class Diagram

The relationship between classes can have multiplicity. The multiplicity is placed near the ends of an association. The multiples indicate the number of instances of one class linked to one instance of the other class. Table 4 shows different multiples.

### Table 4 Multiples

<table>
<thead>
<tr>
<th>Multiplicity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>One</td>
</tr>
<tr>
<td>0..1</td>
<td>Zero or one</td>
</tr>
<tr>
<td>*</td>
<td>Many</td>
</tr>
<tr>
<td>0..*</td>
<td>Between Zero and many</td>
</tr>
<tr>
<td>1..*</td>
<td>Between one and many</td>
</tr>
</tbody>
</table>
A class diagram shows the relations between classes, and how they interact. Classes are shown with attributes and methods. In figure 1, the class diagram for ATM is shown. There are four classes, easily explained the MiniBank Class has the multiple (1, *) relations to CashHandler, this means that 1 MiniBank can have many CashHandlers, but the CashHandler can only be related to one MiniBank. The MiniBank class has a dependency relation to Account, this illustrate that MiniBank class has to have Accounts to be functional.

Figure 1 Example class diagram of a simple ATM

2.4 Sequence Diagrams

The sequence diagrams are a graphical description of objects participating in a use case or scenario. Object can be grouped into swim lanes. The lifeline represents the existence of the object at a particular time. If the object is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the appropriate point, otherwise it goes from the top of the bottom of the diagram.

Notation

The first column corresponds to the actor. The second column should be a boundary object, and the third should be the control object that manages the rest of the scenario.
Actor

The Actor/user in the sequence diagram is the trigger that starts the scenario. An actor is shown as a stitch figure. A user has a goal for using the system, and a given user may have different roles from time to time. The interaction between the actor and the system is what we describe in the sequence diagram. The actor usually starts the sequence of events by asking the system to perform an action. The actor can be human or it could be different system, or a different part of the system. Actor is beside in Sequence diagrams used in Use Case diagrams (having the same function there).

Object

Objects are instances of classes, and are arranged horizontally. The pictorial representation for an Object is a class (a rectangle) with the name prefixed by the object name (optional) and a semi-colon. Object is a data element in an object-oriented system, which has its own identity, belongs to a particular class, and has behavior and properties. The object in the sequence diagrams is a square box with a life line attached. The object is a state but changes state when an event accurse.
Life line

The lifeline is the objects “role”. The lifeline is a dotted line from under the object or the actor. The lifeline represents the existence of the object at a particular time. If the object is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the appropriate point. Usually it’s goes from the top to the bottom of the diagram. The object box is on top of all the lifelines.

When a new object is created during the scenario, then the objects have an object box and lifeline from the creation of the object and to the destruction or end of the diagram

Activation

The Activation shows the focus of control. Activation is shown as a thin square box, covering the lifeline. The activation box shows when the object is active performing an action or are involved in an action. The activation shows the duration of the action in time. There is a controlled relationship between the caller and the activation. The common way to perform a method is by labeling the incoming messages with the indicated action. The top of the activation is at the tip of the incoming message and the base of the activation is on the tip of the return message, or on message to the next action called.

Figure 6: Activation starts with a message from another object, and end with a return message.
Message

A message is a communication between two objects. A message is shown as an arrow between the objects lifelines. In case of a message from the object to itself the arrow may start and finish on the same object symbol. The arrow is labeled with the name of the message, the method call or operation and the argument values. The return message is shown as a dotted arrow.

Figure 7 Messages between objects.
Chapter 3 – Related Work

This chapter describes related research in two distinct areas: The use of UML in an UML-based development process (Section 3.1), and using UML for program comprehension (Section 3.2).

3.1 The Rational Unified Process

The Rational Unified Process (RUP) [14] is a software engineering process that guides a software development organization in their development. The RUP is designed and delivered as a software product. There is need for well-defined and well-documented software development processes. The RUP is like a software tool, designed, delivered and maintained. It is tailored and configured to fit the organization and their needs.

Process Structure: Two Dimensions

The RUP process has two dimensions, as shown in Figure 8. The horizontal axis represents time and the process lifecycle as it unfolds and is the first dimension and it represents the dynamic aspect of the process as it is enacted and expressed in terms of cycle, phases, iterations and milestones.

The vertical axes represent the process workflows and are the second dimension. This is the static aspect of the process. It describes the terms of process components, activities, workflows, artifacts and workers.

Figure 8 Structure of Rational Unified Process
RUP and the UML

RUP is largely about developing and maintaining diagrams. Models are a help to understand and shape both the problem and the solutions. A model is a simplification of reality it helps to manage the large complex systems that can not comprehend in its entirety. UML is the common language to express the different models.

The RUP is a Use Case driven process, the design activities are centered on the notation of the architecture. The main focus of the earlier iterations in RUP is to produce and validate a software architecture that can be an executable prototype. RUP provides a methodical and systematic way to design, develop and validate software and includes templates to describe the architectural views and provides the capture of architectural style, design rules and constraints.

Static Structure: Process Description. A Model of the Rational Unified Process

The RUP describes who is doing what, how and when. RUP is represented using four primary modeling elements:

- WHO the Workers
- HOW the Activities
- WHAT the Artifacts
- WHEN the Workflows

Workers: A worker defines the behavior and responsibilities of the individual or the group. The behavior is expressed in terms of activities that workers perform. Each worker is associated with a set of activities. The reasonability of the worker is usually expressed in relation to certain artifacts that the workers create, modifies and controls.

Activities: Activities are a unit of work that a worker may be asked to perform. The activities have a clear purpose, usually expressed in terms of creating and updating: models, classes and plans.
Artifacts: An artifact is a piece of information that is produced, modified and used by a process. The artifact can be described as the project's sub-products and are used to come to the final product.

Workflows: Workflows are ways to describe meaningful sequences of activities that produce some valuable results and to show interaction between workers. Workflow can be expressed by sequence diagrams, collaboration diagrams, or activities diagrams. There are three types of workflows in RUP: Core workflow, workflow details, and an iteration plan.

Dynamic Structure: Iterative Development
RUP has an iterative development process, the iterative development process goes in circles, there are many small waterfall processes combined in a spiral. A software development process includes four main phases: Requirement analysis, Design, Coding and Testing. This phase's forms iteration, so the iterative process is to address some of the requirement, some of the design and some of the risk. Implement the results and then validate it. Start on the second turn, do a little more requirements, fix the design and design some more, validate it and take another turn. This will continue until the software is finished. The benefits of an iterative process are early risk uncovering, more manageable change, a higher level of reuse, the development is a learning process and the software ends up with better quality, due to the benefits listed.

Making RUP Agile
A software development company, Zuhlke Engineering AG, in Switzerland reports their experience with use of RUP[15].

Project One: Turbine Layout
They lighted the RUP to make it easy to understand and comprehend. In their experience, the RUP proved to be adaptable to the needs of small projects. Their first official RUP project started 1 October 1999. This project had to produce a software tool to design blades for steam turbines that was going to be used by 20-30 mechanical engineers. The development challenges were: short delivery time (9 months), user interface with 2D and 3D, integrating existing software (written by the customer) and vague requirements. Zuhlke meet the challenges with following approach.
A small team of developers, two from the company and one from the customer worked to integrate the existing software. Each iteration lasted a month, this eased the planning and created a steady rhythm. Early and very intense involvement of the customer. Requirement management based on product features, customer wishes was expressed in the product features. Daily build: coded activities already in the first iteration and daily built threw rest of the project. Implementation in Java 2.

The project was completed 2 months ahead of schedule and the customer was satisfied. There was spent more time and effort on the inception and elaboration phases, than on construction. After the delivery the customer came back with some change requests and a few new features, these were implemented in two additional iterations. The project artifacts were decided on at the beginning of the project and documented. The main criteria for the inclusion or exclusion of an artifact were the question: “what value does this artifact add for the customer?” and “what are the likely consequences if we don’t have this artifact?”

Lessons Learned (the Positive Project)

- A sooner completion because of iterative development. The waterfall approach wouldn’t have worked.
- Strong involvement from the customer. A representative from the customer was included in every decision-making.
- Fast and useful feedback from the customer, got feedback after every release, sometimes the customers were shown some of the function in between the releases, this feedback was then included in the next release.
- A small team of experienced and motivated developers.
- Low overhead of project planning, requirements and change management, 7% effort spent on project planning.
- Effective approach to change management.
- The RUP framework and templates saved us a lot of time.

Project two: TV planning system

The second project described in the paper was a TV program planning system. A system that automatically created program schedules for multiple channels and had control of the...
Evaluating the Effect of UML on Maintainability

Cand. Scient Thesis

equipment. The system was going to be implemented as a client-server application. The challenges this time was similar to that of the other project, unclear and changing requirements, tight schedule and the integration of a software component provided by the customer.

**Project Timeline**

The project started: 1 May 2002. The first release was 31 May 2002. By June 2002 the customers had not given any feedback, they did not install the software, so many of the requirements were still unclear. Then there was given different messages from different people in the customer organization. The 2nd release was 31 June 2002, this release was like the previous lacking feedback. The software that was to be integrated with the new system was 2 moths delayed. So there was agreed to create a simulation to this software in the mean time. Another release 31 July that was also ignored, but the release of 30 August 2002 got feedback. The customer wanted a great deal of changes but did not have time to detail them at the desired and needed level. The original task, to make the client, now became secondary. After the release of 30 September, the customer hired a usability expert to make a new GUI for the client. The last release was delivered 31. October 2002, the development team was frustrated and tired. This project was only 70% implemented.

**Lessons Learned**

The customer did not get what they wanted, the project was not completed as Zuhlke wanted and the development team was demoralized. But the project was not a complete failure. The customer got a working, if not the intended piece of software. The main reason for the bad outcome was the lack of feedback, and that key people from the customer were not enough involved. But the customer relations were good, they understood the need of feedback, but did not have the opportunity to provide it. When the project was ended the parting was on good terms. After this project, Zuhlke has successfully completed a dozen more projects with RUP as the development process.

**Suggestion to make RUP agile:**

The Zuhlke team purpose these changes to make a RUP project a success:

1. Select a small set of artifacts. There are only need for 10-12 artifacts for a small project
2. “Must have” artifacts included:
a. A software development plan  
b. A iteration plan  
c. A iteration assessment for each iteration  
d. A software architecture document  
e. A vision document with a list of required features  
f. A change request list  
g. A defect list

3. Iteration planning should focus on results rather than activities. The description of activities and workflow in detail can be viewed as a text book.

On the Rational Edges homepage [16] there are a table suggesting on how to make RUP work, RUP is a framework that needs adjustments to fit into organization. Here is the proposal to success (Table 5):

<table>
<thead>
<tr>
<th>The better way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make it an organizational project and get involvement, even if it is messy</td>
</tr>
<tr>
<td>Review work products and provide input coaching and mentoring allow rework</td>
</tr>
<tr>
<td>Promote process top down, make top management the biggest cheerleaders</td>
</tr>
<tr>
<td>Implement bottom up</td>
</tr>
<tr>
<td>Begin quietly by mentoring individual contributors, and classroom training</td>
</tr>
<tr>
<td>Focus on best practices on addressing everything</td>
</tr>
<tr>
<td>Use RUP phases and milestone to underscore important artifacts</td>
</tr>
</tbody>
</table>

Table 5 Success Factors of the RUP

3.2 Using UML Documents for Program Comprehension

Program comprehension is a complex area because it is hard to identify the way a programmer thinks when he looks at the source code and tries to create a picture of the behavior of the system. Von Mayerhauser and Vans [22] argues that there are two possible approaches of program understanding, viewing program understanding as a top-down approach means that the programmer gains a high level understanding, and than learns the details. While viewing the bottom-up approach of program understanding, the programmers follows the control flow, groups the low-level parts into higher level parts, build a situation model to map parts to code. An experiment [9] was done by Mayerhauser to uncover which of these approaches are usually used during program understanding. They found that there is a switching between the models. The programmer tends to start with the top-down approach,
getting an overview of the code and making assumptions, and then switch to the bottom-up approach to test the high-level assumptions. Program understanding presents both high-level abstractions and low-level details.

In software visualization, graphical models of the system are created. Both static (e.g., structural properties of program code) and dynamic (e.g., program execution traces) information can be visualized. Such graphical representations can, for example, be used to identify and remove performance bottlenecks and to aid in restructuring or program comprehension. For the purposes of program comprehension, the main hypothesis is that graphical representations of a system structure, dataflow and relationships between objects can be of help in software comprehension, and that such information is more valuable than text-based information. However, there is little scientific evidence that graphical documentation is superior to textual documentation (such as the source code, user documentation and textual design documents) for program comprehension.

Many researchers have discussed issues related to which types of diagrams are most appropriate for aiding in program comprehension, however, there is still a lack of empirical data on exactly which forms of graphical documentation are the most suitable for different types of program understanding tasks in specific usage contexts. In the related field of program comprehension, researchers attempt to build models of the underlying mental processes of when developers actually try to understand programs, c.f. [8, 9]. Such knowledge can in turn provide insight into how different kinds of software visualizations might be useful during program comprehension.

Graphical representations of software using the UML notation is often argued to be an effective means for documenting a program to aid in future maintenance tasks. In a recent workshop on program comprehension, the goal was to discuss the usefulness of UML diagrams (as a specific way to visualize programs) to aid in program comprehension [4]. This workshop focused on discussing how much a set of UML diagrams can tell a professional developer (who has some knowledge of the UML) about an existing system that the developer does not know. The UML diagrams might exist from earlier design, or they might be created after a reverse engineering process. To our knowledge, there were no empirical studies presented in this workshop on the effect of UML on program understanding.
Another paper discusses the JAVAVIS system, intended to support the comprehension of Java programs during program execution [6]. The tool monitors a running Java program and visualizes its behavior with UML object and sequence diagrams. The users are able to step through the program line by line or method call by method call and watch the changes in the diagrams. JAVAVIS has been used successfully with students during lectures. The main goal was to shape the thinking about a running Java program and that seemed to work. According to the authors, it was a good decision to use UML diagrams as a program understanding tool.

Agarwal and Sinha [17] have done a study on UML, seen from the developer’s point of view. In their study the subject, how were students, used UML diagrams to model a real-life Object Oriented system. They found that developers usually have positive perceptions of the usability of UML. But when they use the diagrams in real–life situation they don’t experience the diagrams to be so useful after all. When the rating of each diagram was done, the usability was not rated very high. The highest score were received by state diagrams and was only 5.30 out of a possible 7. The authors suggest that the reason can be that there is room for improvement of the diagrams. The diagrams should be simpler and more user friendly. The behavioral diagrams are the ones that need to be more refined, these are the diagrams that show the systems behavior and are the once, one should use to get an overview of what the system are doing.

In the different phases of development, the UML documentation is read differently for different purposes. IBM has done a study addressing this problem [18], and the solution they present are a set of guidelines to follow when the UML diagrams are made, these guidelines are common knowledge, but get be overlooked depending on what the developers purpose of use, for the diagrams. The paper presents a method to keep the technical UML documentation useful. They identify three main actors that use the UML diagrams as documentation, the program developers, technical writers, and graphic designers. The UML is of different use for these different stakeholders. The program developer is made aware of design principles that he might have overlooked, UML shows the technical writer ways to improve the design of the diagram, and UML provides the graphic designer with a methodology to produce a clear, error-free final image that is of manageable size. The goal of IBM was to make smaller viewable diagrams, and give them clear concepts, reduce the translation cost and keep the file size at a minimum.
An experiment was assessed to find the qualititative efficacy of UML diagrams in program understanding [19]. The subjects were asked to analyze a series of UML diagrams and answer a questionnaire for a hypothetical software system. The results suggested that bad design and unstructured UML diagram makes the understanding poorer. The models need to be clear and consistence to give and easy and good understanding of the system at hand. Brewer and Lorenz [20] have used UML to ease the learning of Object Orientation programming. They argue that the student easier understands the programming concepts of object orientation, and get a hold of system thinking by using UML to model there understanding of the problem domain.

UML dynamic behavioral models along with Message Sequence Chart have been used trying to make a graphical presentation format [20] the paper gives a suggestion to a Graphical notation for testing the software. The UML 2.0 Testing Profile defines a way to specify test procedures within UML. It can test the development by using possible information from the system model, in the test model.
Chapter 4 – RESEARCH METHODS

4.1 Empirical Studies

Empirical studies aim at investigating the performance of some technologies and the quality of the resulting object oriented software products. Empirical studies are a method to evaluate the usefulness of the software technology and the effect and improvement potential. Many methods, processes, tools, or notation are being used without satisfactory evaluation. There are two types of approaches to an empirical study the qualitative method and the quantitative research method. There are defined as [23]:

Qualitative research is concerned with studying objects in their natural setting. A qualitative researcher attempts to interpret a phenomena based on explanations that people bring to them. Qualitative research begins with accepting that there is a range of different ways of interpretation. It is concerned with discovering causes noticed by the subjects in the study, and understanding their view of the problem at hand. The subject is the person that is the person, which is taking part in an experiment in order to evaluate an object.

Quantitative research is mainly concerned with quantifying a relationship or to compare two or more groups. The aim is to identify a cause-effect relationship. The quantitative research is often conducted through setting up controlled experiment or collecting data through case studies. Quantitative investigations are appropriate when testing the effect of some manipulation or activity. An advantage is that quantitative data promotes comparisons and statistical analysis.

The main difference between the approaches is to be found in the formulation of the research question. The quantitative research will answer the question with the difference in numbers, while the qualitative research answers will focus on the reason for the differences.

In this thesis there was a mix of the two approaches. During the experiment the subject had to answer a questionnaire as they performed the change tasks. This questionnaire was different for the different treatments. The quantitative data collected were the time used on each task. The qualitative data collect were more of a gathering their opinion on how they performed.
The interviews at the end of the study were done to get a wider view of what they thought of the experiment, when they had given time to reflect over it.

Recent research ranges from identification of Object Oriented features that may cause higher fault rates in software, to a study of effectiveness of different design documentation. If an experiments result is interesting, other researcher can build future on their work, like we have done in our changeability experiment (we build our experiment on the work of Arisholm).

The collection, dissemination and analysis of data are required in most researches. Collecting the data is often the main problem, there are problems attached to maintaining the quality of the data collection. One factor to make the empirical study successful is the state of the data collected, analyzed and presented. That means to ensure reliability, completeness and efficiency of the data. The measurements have to be justified and the underlying assumptions must be made clear to the interpretation of the results. Empirical studies are not always well established beforehand, the research question can be of no interest. The planning does not always cover enough knowledge to verify the investigation of the different underlying issue that the research addresses. Successful empirical research needs a clear defined problem, good data, and a valuable result.

4.2 Method Overview

There are three main types of empirical studies [24]:

Survey: An investigation performed in retrospect, when a tool or technique has been used for a while. The primary means for gathering qualitative and quantitative data are interviews or questionnaire. These are done through taking a sample which is representative from the population being studied. The results from the survey are then analyzed to derive descriptive and exploratory conclusions. They are then generalized to the population from which the sample was taken. A survey provides no control of the execution or the measurements, though it is possible to compare it with similar ones, but it is not possible to manipulate variables as in the other investigations methods.

Case study: to monitor a project, activities or assignments the case study can be used. Data is collected for a specific reason throughout the study. Based on the data collection the statistical analyses can be done. The case studies are normally aimed at tracking a specific attribute or establishing relationship between the different attributes. The level of control is lower for a
case study than for an experiment. A case study is an observational study while the experiment is a controlled study. Case study research is technique where key factor that may have any effect on the outcome are identified and then the activity is documented. Case study is an observational method it is done by observation of an on-going project or activity.

**Controlled Experiment**: Experiments are normally done in a laboratory environment, which provides a high level of control. In an experiment the subjects are assigned to different treatments at random. The objective is to manipulate one or more variable and control all other variable at fixed levels. The effect of the manipulation is measured, and based on this a statistical analysis can be performed. In some cases there may be impossible to use true experimentation, then quasi-experiment may be used. The quasi-experiment is often used when it is impossible to perform random assignment of the subject to the different treatments. An example of an experiment in software engineering is to compare two different types of techniques. For this type of studies methods for statistical inference are applied with the purpose of showing with statistical significance, that one method is better than the other.

An experiment is a formal, rigorous and controlled investigation. In an experiment the key factors are identified and manipulated. The separation between case studies and experiment can be represented by the notation of a state variable. In an experiment the state variable can assume different values and the objective is normally to distinguish between two situations, for example, a control situation and the situation under investigation. Examples of a state variable could be for example, the inspection method or the experience of the software developers. In a case study, the state variable only assumes one value, which is governed by the actual project under study.

We wanted to establish the myth that UML documentation increase the program understanding, and are a tool to get a better platform to maintain the software from. To perform an all qualitative study, like a case study, observing and interviewing could have given us the data desired. This information would probably result in a larger study, which naturally would include an experiment. By combining the experiment and interview we get an empirical support in performing a larger and more complex experiment.
4.3 Controlled Experiments

Experiments are launched when there is need to control the situation, and the behavior will get manipulated directly, precisely and systematically. Many experiments involve more than one treatment to compare the results. An experiment has to be done under controlled condition, and the events are organized in such a way that they match a real world environment.

Characteristics

Experiments are appropriate to investigate different aspects, including:

- Confirm or falsify existing theories
- Confirm conventional wisdom: to test people’s conception
- Explore relationships or test that a certain relationship holds
- Evaluate the accuracy of models: to test the accuracy of certain models is as expected.
- Validate measures: to ensure that a measure actually measures what is supposed to

The strength of experiments are that there can be investigated in which situation the claims are true, and they provide a context in which certain standards, methods and tools are recommended for use.

Experiment Process

Conducting an experiment involves several steps. The different steps are:

- Definition: formulating the hypotheses
- Planning: finding the population to be tested and preparing the experiment materiel
- Execution: the actual execution of the experiment with the subjects
- Analysis and interpretation
- Presentation and package

Experimentation is not simple there is a lot work included, preparation, conduction and then analyzing the experiment properly. The main advantage of an experiment is the control of the subjects, objects and the experiment condition. Other advantage is that there can be performed statistical analysis and there is an opportunity to replicate.

Design of Experiments

Many factors should be considered when the experiment is designed.
Proper materials like code listings need to be readily available and tested to ensure that minor problems are identified before the main experiment. A pilot study can reveal several features that could undermine the result.

Select the subject from a population that can perform the tasks that the experiment contains of, or they should be trained in the task solving problematic.

The groups should be comparable in terms of experience. This is important because individual capability can be an overriding factor in software development.

The experiment should be repeatable, and the material should be available to whole community. A replication package should be available containing details of the experiment.

The subject should get all the information it need on the computer.

The research performed in this thesis, was focused finding out how the programmer thinks and what kind of document can be of help, when a programmer is changing and maintaining a software system. The best way to get answers to this question is to perform an experiment.

The experiment that is conducted, in this thesis, had taken into consideration the key elements of the experiment design, the experiment was a replicate of Erik [2] experiment on changeability on Mainframe vs. Responsibility driven design. The focus in this experiment was No UML vs. UML, and UML documentation was added. The subjects all had Java and UML experience, they were randomly assigned. All the material was on the computer, and they could print anything they wanted on paper.
Chapter 5 - The UML experiment

5.1 Experiment Design

The experiment consisted of three phases, in which both quantitative and qualitative data pertaining to the subjects experience and their performance in the experiment was collected.

In the first phase, the subjects filled out an experience questionnaire, containing questions regarding education, programming experience, UML experience, Tau UML experience, work experience and so on (Appendix C). After this, the subjects were assigned to different experimental treatments using a randomized blocking scheme.

The second phase was the actual experiment. The subjects had to perform five change tasks within eight hours. The Simula Experiment Support Environment (SESE) was used to download task descriptions and code, upload task solutions and answer task questionnaires [3]. Details of the tasks and questionnaires are given in Appendix D. The subjects could leave when they finished all the tasks. One subject left after 2½ hours, but most did not leave until after 6 or 7 hours. Many subjects stayed until the end of the experiment.

One week after the main phase, individual interviews were conducted with the subjects. These interviews provided additional, qualitative data on the experiment. The interview guide is given in Appendix B.

5.2 Hypotheses

The experiment attempted to test the following three (null) hypotheses:

H1: Coding Effort (H₀): There is no difference in the time required to understand, code and test the Java code in object oriented systems that are unknown to the developer if the system is specified with UML diagrams than if no such design information is available.

H2: Total Effort (H₀): There is no difference between the total time required to understand, code and test the Java code and update the UML diagrams (for those who received UML...
Evaluating the Effect of UML on Maintainability and the time required to understand, code and test the Java code (for those who did not receive UML documentation).

H3: Correctness (H0): The probability of providing correct solutions for a set of given change tasks on an object oriented system unknown to the developer is higher if the system is provided with UML design diagrams than if no such design information is available.

5.3 Treatments

The treatments of the experiment contained a total of five change tasks on two Java systems, an Automated Teller Machine (ATM) and a Coffee Machine, respectively. The treatments were also used in previous studies [1, 2]. The experiments reported in [1, 2] used two alternative designs of the coffee-machine with a centralized and delegated control style, respectively. The results from the previous studies indicated that the coffee-machine design with a delegated control style was very difficult to understand and change correctly, in particular for inexperienced developers such as undergraduate students [1]. Consequently, to assess the benefits of UML, the coffee-machine with a delegated control style was chosen as the treatments for this experiment (in addition to the ATM system).

Change Tasks

The subjects had to perform five change tasks (in addition to a small training task):

Task 1 (ATM): Add functionality on the ATM to print out an account transaction statement.
Task 2 (Coffee Machine): Implement a coin return-button.
Task 3 (Coffee Machine): Make bouillon as a new type of drink.
Task 4 (Coffee Machine): Check whether all ingredients are available for the selected drink.
Task 5 (Coffee Machine): Make your own drink by selecting among the available ingredients.

All subjects used Emacs and a Java compiler to perform the change tasks. The complete task descriptions are provided in Appendix F.

UML Documentation

Complete UML documentation was developed for the ATM and the delegated control-style coffee machine design. The UML designs included a class diagram and a complete set of documentation.
sequence diagrams for both systems (details are provided in appendix E). The sequence and class diagrams were made in Tau UML [7].

The UML documents adhere to the notation guidelines given in [7] as much as possible. However, Tau UML does not support all of the available UML notations. In particular, Tau UML can only show one level of activation boxes in sequence diagrams, whereas UML allows for several levels of activation boxes to indicate nested method calls. Thus, since we lacked tool support for second level activation boxes, we created notes in the sequence diagrams that indicated where a second level activation box would start and where it would end. In addition, we decided that every sequence diagram should include a complete sequence of user events from the start of the program. Thus, each sequence diagram shows one particular variation of a complete sequence of events. The reason for this was that it made it less complex to read and the subject could easier see how a particular sequence of events was initiated.

5.4 Group Assignments

<table>
<thead>
<tr>
<th></th>
<th>UML</th>
<th>No UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Think-aloud</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Silent condition</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 6. Group assignment of the experiment

The subjects were divided in two main groups, UML and No UML (Table 1). The Java programming tasks were identical for both groups, but the subjects assigned to the UML condition used the case tool Tau-UML to read and update UML design documentation for each Java program, whereas the subjects assigned to the No UML received no UML documentation.

In addition, the two main groups where further divided into two sub-groups, Think-aloud and Silent condition (Table 1). For the subjects assigned to the Think-aloud condition, the think-aloud screen appeared every 20 minutes with the text 'How have you been thinking while solving this task?' The subjects were instructed to recall their thoughts while solving the last task on the think-aloud screen. The available time for writing comments in the think-aloud screen was limited to two minutes. The comments from the think-aloud screen were used to
study how the UML documentation changed the way in which they understood, coded and tested the change tasks.

The subjects were assigned to the groups by means of randomization and blocking. The blocking criteria were the amount of credits earned in Informatics. (Of UML and TAU_UML experience (Appendix A))

5.5 Data Collection

Quantitative Data Collection
The main dependent variables of the experiment were the time spent to solve each task and the correctness of the code solutions of each task. The time data was reported by the subjects using a change task questionnaire provided to the subjects through the SESE tool (Appendix D). The correctness of each solution was assessed by one of the authors of this paper. Each task was deemed *correct* if the solution contained no serious logical errors (minor cosmetic differences compared with the test-cases provided with the task descriptions were allowed), otherwise the solution was *incorrect*.

Qualitative Data Collection
Qualitative data were collected from three sources: Interviews, change task questionnaire comments, and think-aloud comments.

*Interviews*
After the main phase of the experiment, semi-structured interviews were conducted with the subjects. An interview guide with relatively open questions was prepared (Appendix B). The interviews were recorded on tape. 19 interviews were conducted within one week. They lasted about 12 to 35 minutes, depending on what type of condition the subjects were assigned to and on how talkative the subjects were. During the interviews, shortcomings in the interview guide were found. When the guide was designed, the assumption was that those assigned to the UML condition used the documentation in order to understand the program and several questions were related to this. When it became evident that several did not use the documentation in the expected way, questions related to this became redundant. Consequently, mid-way in the interview schedule, the interview guide was extended.
Task Comments
The subjects completed a questionnaire after each task. Amongst others, the questionnaire contained a free-text field in which the subjects could report anything they thought might be relevant in explaining the results (e.g., time spent) on each task. The content of the questionnaires were stored in a database during the experiment. 76 out of a total of 120 comments were filled out.

Think Aloud Comments
Half of the subjects were assigned to the think aloud condition. 225 comments were collected and stored in a database during the experiment.

Analysis - Quantitative Data Analysis
To test the hypotheses of the experiment, a regression based approach was considered. However, due to the limited number of subjects in the experiment, there was insufficient statistical power to achieve reliable results with such tests. Thus, only descriptive statistics and box-plots are provided, to informally test the hypotheses.

Analysis - Qualitative Data Analysis
Interviews
To increase the accuracy and comprehensiveness of the analysis, the recorded interviews were transcribed word by word. The interview transcripts were analyzed using QRS N6, a tool for qualitative data analysis. Using the tool, the interviews were examined, sorted and categorized into different concepts and a tree structure with concepts and sub-concepts was build. The analysis was divided into two separate parts:

- UML; about the use and effect of UML diagrams, and
- Think aloud, about the use and effect of the think aloud screen.

The analysis of the use and effect of the think aloud screen is out of the scope of this report, but can be found in [5]. In the qualitative analysis of concerns relating to UML, the subjects were divided into two groups; subjects assigned to UML condition and subjects assigned to the No UML condition, respectively. In the analysis, the categorization of the subject’s answers were both within and crosswise the two groups. The analysis was conducted independently of the quantitative analysis, that is, by different researchers.
Questionnaire Comments
The qualitative comments from the questionnaire were retrieved from the experiment database and formatted into text files. As with the interviews, the comments were also structured and examined with QRS N6. A tree structure with categories and sub-categories was built. The comments from each task were first gathered and then categorized. The comments belonging to each task were compared. Furthermore, the comments were analyzed with respect to the same groups as in the interview analysis.

Think Aloud Comments
The think aloud comments were retrieved from the experiment database, formatted and finally analyzed supported by QRS N6. A tree structure with categories and sub-categories was built and the comments were examined and analyzed with respect to the two subject groups.

5.6 Results
Quantitative Results
Table 7 shows the descriptive statistics of the effort spent to solve the change tasks for the different groups, related to hypothesis $H1$. The results indicate that the subjects receiving UML documentation spent on average 25 percent less time to solve the tasks than did the subjects without UML documentation. Furthermore, the variance is much lower for the subjects who received UML documentation. A boxplot of the total coding effort data are provided in Figure 9.
### Table 7. Descriptive statistics of effort (in minutes) to understand, code and test the change tasks (H1)

<table>
<thead>
<tr>
<th>Task</th>
<th>Group</th>
<th>N</th>
<th>Mean</th>
<th>Median</th>
<th>StDev</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Q1</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>No UML</td>
<td>11</td>
<td>84</td>
<td>75</td>
<td>60</td>
<td>240</td>
<td>43</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>9</td>
<td>54</td>
<td>53</td>
<td>19</td>
<td>31</td>
<td>95</td>
<td>38</td>
<td>64</td>
</tr>
<tr>
<td>Task 2</td>
<td>No UML</td>
<td>11</td>
<td>23</td>
<td>20</td>
<td>16</td>
<td>5</td>
<td>60</td>
<td>13</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>9</td>
<td>16</td>
<td>15</td>
<td>5</td>
<td>8</td>
<td>23</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>Task 3</td>
<td>No UML</td>
<td>11</td>
<td>24</td>
<td>22</td>
<td>13</td>
<td>10</td>
<td>55</td>
<td>14</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>9</td>
<td>21</td>
<td>19</td>
<td>7</td>
<td>12</td>
<td>35</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>Task 4</td>
<td>No UML</td>
<td>11</td>
<td>78</td>
<td>54</td>
<td>46</td>
<td>16</td>
<td>150</td>
<td>44</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>9</td>
<td>65</td>
<td>65</td>
<td>16</td>
<td>45</td>
<td>95</td>
<td>52</td>
<td>75</td>
</tr>
<tr>
<td>All Tasks</td>
<td>No UML</td>
<td>11</td>
<td>209</td>
<td>195</td>
<td>98</td>
<td>51</td>
<td>363</td>
<td>132</td>
<td>277</td>
</tr>
<tr>
<td></td>
<td>UML</td>
<td>9</td>
<td>157</td>
<td>163</td>
<td>30</td>
<td>98</td>
<td>189</td>
<td>136</td>
<td>184</td>
</tr>
</tbody>
</table>

**Figure 9.** Boxplot of coding effort versus UML/No UML
Table 8. Descriptive statistics of total effort (in minutes) to understand, code and test the change tasks and update UML documentation (H2)

Table 8 shows the descriptive statistics of the effort spent, but including the time spent to update the UML documentation, related to hypothesis $H2$. The results suggest that having to update the UML documents incurs an effort overhead. Thus, while the results suggest that the effort spent to actually understand, code and test the solutions are less if UML documentation is available, the results in Table 3 suggest that the additional effort required to update the UML documentation is larger than the time saved by having better documentation.

Finally, Figure 10 gives a graphical comparison of the percentage of subjects with correct solutions for the two groups. Overall, the subjects receiving UML documentation seem to have a better chance of producing correct solutions. However, the benefits are quite small for the simpler tasks (task 1-3) and very large for the more complex task (task 4). In contrast to the other tasks, Task 4 requires a very deep understanding of how the coffee-machine operates to solve the task correctly. In such cases, having access to the UML documentation seems to be very beneficial.
Figure 10. Percent of subjects with correct solutions on the different tasks

Qualitative Results

The analysis results of the interviews, the questionnaire comments and the think aloud comments are presented in Table 9. The results suggest that the use and benefit of UML documentation varied among the subjects assigned to the UML condition. The experiment required that all the subjects had to update the diagrams before they moved on to the next task. However, whether they used the documentation to identify change locations prior to performing code modifications or they ignored it varied among the subjects. Some of those who used the diagrams said they were more useful on the complicated task and that they often were superfluous on the easy tasks.
The subjects that claimed not to have used the documentation explained the following reasons for not using it:

- Habit; they were more used to look at the code than diagrams.
- Time pressure; they felt that to study the documentation required too much time.
- They did not use it from the start; consequently it became difficult to use it later
- The tasks were relatively small; therefore it was easier to study the code.

Almost all made the code changes first and thereafter updated the documentation. Some complained about the updating of the diagrams because the diagrams were untidy or difficult to update. Several expressed dissatisfaction with TAU.

Many of the subjects who were assigned to the No UML condition said that it was difficult to get an overview of the programs. This was especially evident in the questionnaire comments, but also to a certain extent in the interviews and the think aloud comments. Some subjects drew their own diagrams when they solved the difficult tasks. Several thought that UML diagrams would have helped them to locate where to do the code changes, and additionally, some claimed that it would have helped them with how to solve the tasks. Furthermore, some mentioned that UML documentation would have helped them to make better solutions. UML was considered to be most useful on the complex tasks and not so useful on the easy tasks.
<table>
<thead>
<tr>
<th>TOPICS</th>
<th>UML</th>
<th>No UML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The use of UML documentation</td>
<td>• The UML documentation was used to locate code changes (2/9)</td>
<td>• UML diagrams were drawn by hand to help solving the most difficult tasks (4/10)</td>
</tr>
<tr>
<td></td>
<td>• The documentation was used a little prior to code changes (3/9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The documentation was ignored prior to code changes (4/9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The documentation was updated after the code changes (7/9)</td>
<td></td>
</tr>
<tr>
<td>The benefit of UML documentation</td>
<td>• UML useful for solving the tasks by providing overview and structure (2/9)</td>
<td>• UML would have been useful to locate the code changes (8/10)</td>
</tr>
<tr>
<td></td>
<td>• UML useful for control of code changes and overview to some degree (3/9)</td>
<td>• UML would have been helpful with how to do the code changes (4/10)</td>
</tr>
<tr>
<td></td>
<td>• UML not useful for the task solving (4/9)</td>
<td>• UML would not have been useful on the easiest tasks (15/20*)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UML documentation would have improved the solution (3/10)</td>
</tr>
<tr>
<td>Time</td>
<td>• Time pressure at the end of the experiment (8/9)</td>
<td>• Time pressure at the end of the experiment (4/10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• There was sufficient time for task solving (6/10)</td>
</tr>
<tr>
<td>Problems</td>
<td>• It was bothersome and difficult to update the UML diagrams (3/9)</td>
<td>• Java-related problems in the beginning of the experiment (4/6***)</td>
</tr>
<tr>
<td></td>
<td>• Dissatisfaction with Tau_UML (6/9)</td>
<td>• Out of practice with programming/Java (7/10)</td>
</tr>
<tr>
<td></td>
<td>• Java-related problems in the beginning of the experiment (2/4***)</td>
<td>• Problems with maintaining the overview of the program (10**)</td>
</tr>
<tr>
<td></td>
<td>• Problems with maintaining the overview of the last task (4/9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Out of practice with Java programming (2/9)</td>
<td></td>
</tr>
</tbody>
</table>

Table 9. Summary of qualitative results (the numbers in parentheses indicate the proportion of subjects for which the statement applies).  
( * = the statement is a combination of two questions  
** = the statement is based on questionnaire comments  
*** = the statement is based on think aloud comments)

In the questionnaire comments and in the think aloud comments, the subjects assigned to no UML wrote more often that they did not complete the tasks than the UML subjects. Furthermore, according to the think aloud comments, slightly more Java related problems on the two first tasks could be observed among those who did not have UML. Some of the UML subjects thought it was difficult to maintain an overview of the program in relation to the last task. However, this comment was a lot more widespread among the No UML subjects, who had this problem in relation to several tasks.
The quantitative results described in Section 5.6 suggest that the UML subjects were more likely to produce correct solutions than the No UML subjects, especially on Task 4. According to Table 3, a common feature of the No UML subjects was that they found it difficult to get an overview of the program. These types of comments were not so widespread among the UML subjects. In fact, half of the UML subjects considered UML useful for obtaining an overview of the code, although the degree of perceived benefit varied considerably among these. Consequently, it is possible to presume that by giving overview of a complex program, the UML documentation provide an advantage to those assigned to the UML condition.

Interestingly, Table 8 also shows that about half of the UML subjects claimed that the UML was not useful for them and that they did not use it beyond the required update. Thus, the subjective opinions expressed by the subjects contradict the quantitative results, which clearly suggest benefits both in terms of effort and correctness!

### 5.7 Threats to Validity

#### Construct Validity

In this experiment, both quantitative and qualitative data was collected in an attempt to provide more reliable (triangulated) results. This section identifies weaknesses and limitations of the different data sources.

#### Quantitative Data Collection

Some subjects mentioned that it was not clear whether they could use Java API or other Java help sources. Some said that they spent much time and effort on Java specific difficulties that easily could have been solved if they could have used help sources. If some used help sources and others did not, the subjects did not have the same basis for solving the tasks. The effort spent on each task and the correctness of the solutions may be influenced by this.

#### Qualitative Data Collection

The three qualitative data collection techniques used in this experiment; interviews, questionnaire comments and think aloud comments, provide different types of information at different levels.
Interviews
In the interviews, the information obtained was related to the experiment in general. The interviews were also suitable to provide information that could explain motivation and reasons for different actions. In addition, information about different tasks was retrieved, but seldom as detailed as with the questionnaire comments and in particular, the think aloud comments. By interviewing the subjects, information from everybody was obtained. However, as for the other qualitative data sources, the information obtained only reflect the subjective views of the subjects.

Questionnaire Comments
The questionnaires comments are, similar to the think aloud comments, related to the current task. However, they are more general and provide an overview of the task. These comments allow the subjects to comment their solutions and explain what problems they spent time on. The comments are useful for identifying the main problems of the task just completed. These comments can be regarded as higher level comments than the think aloud comments. Similar to the think aloud comments, they are not very explanatory about motivations and reasons for actions. The information retrieved is very dependent on how much and how thoroughly the subjects write. About 37 % of the comment fields were empty, thus they do not necessarily provide a complete picture of problems experienced during the experiment.

Think Aloud Comments
The think aloud comments are of an immediate kind; they provide information about what the subjects feel and think when the screen appears. The information obtained depends on the subjects’ personal style. It can be very code and task specific, or more general about feelings or the experiment situation. However, the comments are very dependent on how well the subjects write and how willing they are to share their thoughts in the think aloud screen. The comments do rarely explain motivations and reasons for actions.

Internal Validity
This experiment involved only 20 subjects. Thus, it was difficult to ensure that the subjects assigned to the two groups had similar programming skills. The fact that slightly more Java-related problems could be observed among the No UML subjects and additionally, that several said that they were out of practice with programming/Java, makes it possible that subjects assigned to the UML condition had better programming skills than did the subjects assigned to
the No UML condition. Consequently, the use of UML documentation might not be the only reason to why the UML subjects performed better.

External Validity

The programs and tasks of this experiment were small in size and duration. It is likely that the effect of UML on change effort and correctness would have been larger for larger and more complex tasks. Furthermore, only students with limited programming skill and experience with maintenance tasks participated. Thus, the results are difficult to generalize to the population we ideally would like to make claims about, that is, more experienced, professional developers solving larger programming tasks. Finally, the results regarding the effort to update the UML documentation is likely to be quite tool dependent: it is possible that using other tools than Tau UML would have reduced the effort to update the diagrams.
Chapter 6 – Conclusions and Future Research

Software maintenance cost are high, and to address this problem there are many different tradeoffs. The better the knowledge of the system, the easier it is to perform changes. The hypotheses underlying the experiment described in this thesis is that with UML documentation, the costs of making changes are reduced and the changes are more correct than when done without any kind of design documentation. The results indicate that having access to UML documentation increases programmer productivity and program quality when maintaining systems for which the developers have no prior knowledge. However, having to update the UML documentation also incurs additional costs.

This tradeoff between costs and benefits need to be studied further, as this experiment is unable to assess the long-term effects of increased programming productivity and program quality versus the UML documentation maintenance overhead. Thus, although the results are promising, more studies are required to further assess the costs and benefits of using UML during program maintenance. In future controlled experiments, several threats to validity should also be addressed: The number of subjects should be increased to ensure that the results are not biased due to differences in skills between the subjects. Professional developers should participate. The experiment should be replicated with other UML tools. Finally, the size of the programs and tasks should be increased.
References


Evaluating the Effect of UML on Maintainability


[16] http://therationaledge.com/content/jun_03/f_topfive_bb.jsp


Erik Arisholm, Samera Afsheen Ali, Siw Elisabeth Hove “An Initial Controlled Experiment to Evaluate the Effect of UML Design Documentation on the Maintainability of Object-Oriented Software in a Realistic Programming Environment”
# Appendix A – Experience Data of the Subjects

<table>
<thead>
<tr>
<th>Subject</th>
<th>Programming experience in work</th>
<th>Total work experience</th>
<th>Credits in programming</th>
<th>Credits in total</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>409</td>
<td>0</td>
<td>5</td>
<td>16</td>
<td>48</td>
</tr>
<tr>
<td>410</td>
<td>0</td>
<td>3</td>
<td>24</td>
<td>57</td>
</tr>
<tr>
<td>416</td>
<td>0</td>
<td>2</td>
<td>17</td>
<td>67</td>
</tr>
<tr>
<td>420</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>67</td>
</tr>
<tr>
<td>425</td>
<td>0</td>
<td>3</td>
<td>20</td>
<td>61</td>
</tr>
<tr>
<td>402</td>
<td>0</td>
<td>4</td>
<td>24</td>
<td>102</td>
</tr>
<tr>
<td>405</td>
<td>0</td>
<td>2</td>
<td>20</td>
<td>71</td>
</tr>
<tr>
<td>407</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>41</td>
</tr>
<tr>
<td>414</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>51</td>
</tr>
<tr>
<td>417</td>
<td>0</td>
<td>1</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>423</td>
<td>1</td>
<td>1</td>
<td>20</td>
<td>66</td>
</tr>
<tr>
<td>401</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>70</td>
</tr>
<tr>
<td>403</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>404</td>
<td>0</td>
<td>5</td>
<td>40</td>
<td>77</td>
</tr>
<tr>
<td>411</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>66</td>
</tr>
<tr>
<td>413</td>
<td>0</td>
<td>1</td>
<td>30</td>
<td>78</td>
</tr>
<tr>
<td>424</td>
<td>0</td>
<td>7</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>408</td>
<td>0</td>
<td>3</td>
<td>30</td>
<td>49</td>
</tr>
<tr>
<td>415</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>58</td>
</tr>
<tr>
<td>418</td>
<td>0</td>
<td>2</td>
<td>30</td>
<td>98</td>
</tr>
<tr>
<td>419</td>
<td>3</td>
<td>8</td>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>422</td>
<td>3</td>
<td>3</td>
<td>30</td>
<td>140</td>
</tr>
<tr>
<td>PDF1T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>409</td>
<td>2</td>
<td>3</td>
<td>2400</td>
<td>1</td>
</tr>
<tr>
<td>410</td>
<td>2</td>
<td>3</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>416</td>
<td>3</td>
<td>3</td>
<td>2000</td>
<td>1</td>
</tr>
<tr>
<td>420</td>
<td>3</td>
<td>3</td>
<td>5000</td>
<td>1</td>
</tr>
<tr>
<td>425</td>
<td>3</td>
<td>4</td>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>402</td>
<td>3</td>
<td>4</td>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>405</td>
<td>2</td>
<td>2</td>
<td>4000</td>
<td>1</td>
</tr>
<tr>
<td>407</td>
<td>3</td>
<td>3</td>
<td>8000</td>
<td>1</td>
</tr>
<tr>
<td>414</td>
<td>3</td>
<td>3</td>
<td>5000</td>
<td>1</td>
</tr>
<tr>
<td>417</td>
<td>2</td>
<td>2</td>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>423</td>
<td>3</td>
<td>4</td>
<td>5000</td>
<td>1</td>
</tr>
<tr>
<td>PDF2T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>401</td>
<td>2</td>
<td>4</td>
<td>4000</td>
<td>3</td>
</tr>
<tr>
<td>403</td>
<td>3</td>
<td>2</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>404</td>
<td>4</td>
<td>4</td>
<td>6500</td>
<td>2</td>
</tr>
<tr>
<td>411</td>
<td>3</td>
<td>3</td>
<td>4000</td>
<td>2</td>
</tr>
<tr>
<td>413</td>
<td>4</td>
<td>4</td>
<td>20000</td>
<td>1</td>
</tr>
<tr>
<td>424</td>
<td>2</td>
<td>3</td>
<td>2000</td>
<td>1</td>
</tr>
<tr>
<td>PDF2T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>408</td>
<td>3</td>
<td>4</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>415</td>
<td>3</td>
<td>3</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>418</td>
<td>4</td>
<td>3</td>
<td>8000</td>
<td>1</td>
</tr>
<tr>
<td>419</td>
<td>4</td>
<td>3</td>
<td>10000</td>
<td>1</td>
</tr>
<tr>
<td>422</td>
<td>3</td>
<td>4</td>
<td>50000</td>
<td>2</td>
</tr>
</tbody>
</table>
Evaluating the Effect of UML on Maintainability

### Small Talk

<table>
<thead>
<tr>
<th>Subject</th>
<th>Lines in Small talk</th>
<th>Compt Lines in C</th>
<th>Comp Pascal Lines in Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF1</td>
<td>600</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>409</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>410</td>
<td>2</td>
<td>4000</td>
<td>1</td>
</tr>
<tr>
<td>416</td>
<td>2</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>420</td>
<td>2</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>425</td>
<td>3</td>
<td>800</td>
<td>1</td>
</tr>
<tr>
<td>402</td>
<td>3</td>
<td>1500</td>
<td>1</td>
</tr>
<tr>
<td>405</td>
<td>1</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>407</td>
<td>3</td>
<td>3500</td>
<td>1</td>
</tr>
<tr>
<td>414</td>
<td>2</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>417</td>
<td>2</td>
<td>1500</td>
<td>1</td>
</tr>
<tr>
<td>423</td>
<td>1</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>PDF1T</td>
<td>403</td>
<td>4</td>
<td>7000</td>
</tr>
<tr>
<td>404</td>
<td>3</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>411</td>
<td>3</td>
<td>5000</td>
<td>2</td>
</tr>
<tr>
<td>413</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>424</td>
<td>3</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>PDF2</td>
<td>408</td>
<td>3</td>
<td>2000</td>
</tr>
<tr>
<td>415</td>
<td>2</td>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>418</td>
<td>2</td>
<td>2000</td>
<td>2</td>
</tr>
<tr>
<td>419</td>
<td>2</td>
<td>2000</td>
<td>1</td>
</tr>
<tr>
<td>422</td>
<td>2</td>
<td>5000</td>
<td>2</td>
</tr>
</tbody>
</table>

### Other program language

<table>
<thead>
<tr>
<th>Subject</th>
<th>Compt in this language</th>
<th>Lines</th>
<th>Other program language Compt</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF1</td>
<td>Perl</td>
<td>2</td>
<td>Python</td>
<td>2</td>
</tr>
<tr>
<td>410</td>
<td>500</td>
<td></td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>416</td>
<td>40</td>
<td>2</td>
<td>ML</td>
<td>30</td>
</tr>
<tr>
<td>420</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDF1T</td>
<td>html</td>
<td>4</td>
<td>php</td>
<td>2</td>
</tr>
<tr>
<td>402</td>
<td>1500</td>
<td></td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>405</td>
<td>perl</td>
<td>3</td>
<td>python</td>
<td>2</td>
</tr>
<tr>
<td>407</td>
<td>Python</td>
<td>3</td>
<td>Perl</td>
<td>300</td>
</tr>
<tr>
<td>414</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>417</td>
<td>Some</td>
<td>2</td>
<td>Maude</td>
<td></td>
</tr>
<tr>
<td>423</td>
<td>Perl</td>
<td>3</td>
<td>Python</td>
<td>500</td>
</tr>
<tr>
<td>PDF2</td>
<td>html</td>
<td>5</td>
<td>php</td>
<td>3</td>
</tr>
<tr>
<td>401</td>
<td>20000</td>
<td></td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>404</td>
<td>Perl</td>
<td>4</td>
<td>Python</td>
<td>3</td>
</tr>
<tr>
<td>411</td>
<td>2000</td>
<td></td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>413</td>
<td>Perl</td>
<td>5</td>
<td>Cobol</td>
<td>2</td>
</tr>
<tr>
<td>424</td>
<td>some</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDF2T</td>
<td>Python</td>
<td>2</td>
<td>ML</td>
<td>1</td>
</tr>
<tr>
<td>408</td>
<td>100</td>
<td></td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>415</td>
<td>Maude</td>
<td>2</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>418</td>
<td>Perl</td>
<td>4</td>
<td>2500</td>
<td></td>
</tr>
<tr>
<td>419</td>
<td>PHP</td>
<td>5</td>
<td>LISP</td>
<td>2</td>
</tr>
<tr>
<td>422</td>
<td>VB</td>
<td>2</td>
<td>Prolog</td>
<td>3000</td>
</tr>
</tbody>
</table>

| Deleted: Maintanability |
|-------------------------|-----------------|-----------------|-----------------|-----------------|
| Deleted:                |
| Deleted:                |
| Deleted: t              |
## Design experience

<table>
<thead>
<tr>
<th>Subject</th>
<th>Compt UML</th>
<th>Compt tau.uml</th>
<th>Compt Rational</th>
<th>Compt Rose</th>
<th>Compt OMT</th>
<th>Compt RD</th>
<th>Compt CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF1</td>
<td>409</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>410</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>416</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>420</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>425</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PDF1T</td>
<td>402</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>405</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>407</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>414</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>417</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>423</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>PDF2</td>
<td>401</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>403</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>411</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>413</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>424</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>PDF2T</td>
<td>408</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>415</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>418</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>419</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>422</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Compt Role modeling</th>
<th>Structure design and analysis</th>
<th>Compt Data driven design</th>
<th>Comp other design methods</th>
<th>Comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDF1</td>
<td>409</td>
<td>2</td>
<td>2</td>
<td>OOA&amp;D(in265)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>410</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>416</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>420</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>425</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDF1T</td>
<td>402</td>
<td>1</td>
<td>1</td>
<td>use case driven design</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>405</td>
<td>1</td>
<td>2</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>407</td>
<td>3</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>414</td>
<td>3</td>
<td>1</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>417</td>
<td>1</td>
<td>3</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>423</td>
<td>3</td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>PDF2</td>
<td>401</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>411</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>413</td>
<td>3</td>
<td>3</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>424</td>
<td>4</td>
<td>2</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>PDF2T</td>
<td>408</td>
<td>3</td>
<td>3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>415</td>
<td>3</td>
<td>2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>419</td>
<td>2</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>422</td>
<td>1</td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

*Specification and Description Language*
Appendix B – Interview Questionnaires

Intervjuguide 1

Oppgavene
1. [ALLE] Hva synes du generelt om eksperimentet? (Fortell kort om hvordan du opplevde eksperimentet, tidspress, stress?)
2. [ALLE] Hvordan var informationsmaterialet dere fikk? (Skriftlig informasjon i starten av eksperimentet, informasjon igjennom SESE, uklarheter etc?)
3. [ALLE] Hvordan var oppgaveformuleringene?
4. [ALLE] Var du usikker på hva du skulle gjøre, og hadde det i så fall hjulpet med bedre instruksjoner?
5. [ALLE] Hvordan opplevde du å bruke verktøyene (TAU og SESE) under eksperimentet? (Oppstod det vanskeligheter pga verktøyene? Hvilke konsekvenser hadde det for deg?)

6. [ALLE] Var det en bestemt oppgave du syntes var spesielt enkel?
   1. [ALLE] Hvilken oppgave var det?
   2. [ALLE] Hva gjorde at den var enkel?

7. [ALLE] Var det en bestemt oppgave du syntes var spesielt vanskelig?
   1. [ALLE] Hvilken oppgave var det?
   2. [ALLE] Hva gjorde at den var vanskelig?

8. For den oppgaven du synes var vanskeligst:
   1. [UML] I hvilken grad synes du UML dokumentasjonen var nyttig for å løse oppgaven? På hvilken måte?
   2. [ALLE] Hva synes du var vanskeligst: Å finne ut HVOR du skulle gjøre endringene eller HVORDAN du skulle gjøre endringene?
   3. [ALLE] Hva synes du var vanskeligst: Å finne ut HVOR/HVORDAN du skulle gjøre endringene, eller selve Java kodingen?
   4. [UML] Hva synes du var vanskeligst: Å oppdatere UML diagrammene eller selve Java kodingen?
   5. [UML] I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVOR du skulle gjøre endringene?
   6. [IKKE UML] I hvilken grad tror du det hadde vært lettere å forstå HVOR du skulle gjøre endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og sekvensdiagrammer)?
   7. [UML] I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVORDAN du skulle gjøre endringene?
   8. [IKKE UML] I hvilken grad tror du det hadde vært lettere å forstå HVORDAN du skulle gjøre endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og sekvensdiagrammer)?
9. For den oppgaven du synes var enklest:
   1. [UML] I hvilken grad synes du UML dokumentasjonen var nyttig for å løse oppgaven? På hvilken måte?
   2. [ALLE] Hva synes du var vanskeligst: Å finne ut HVOR du skulle gjøre endringene eller HVORDAN du skulle gjøre endringene?
   3. [ALLE] Hva synes du var vanskeligst: Å finne ut HVOR/HVORDAN du skulle gjøre endringene, eller selve Java kodingen?
   4. [UML] Hva synes du var vanskeligst: Å oppdatere UML diagrammene eller selve Java kodingen?
   5. [UML] I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVOR du skulle gjøre endringene?
   6. [IKKE UML] I hvilken grad tror du det hadde vært lettere å forstå HVOR du skulle gjøre endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og sekvensdiagrammer)?
   7. [UML] I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVORDAN du skulle gjøre endringene?
   8. [IKKE UML] I hvilken grad tror du det hadde vært lettere å forstå HVORDAN du skulle gjøre endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og sekvensdiagrammer)?

\textbf{Intervjuguide 2}

\textit{Oppgavene}

1. [ALLE] Hva synes du generelt om eksperimentet? (Fortell kort om hvordan du opplevde eksperimentet, tidspress, stress?)
2. [ALLE] Hvordan var informasjonsmaterialet dere fikk? (Skriftlig informasjon i starten av eksperimentet, informasjon igjennom SESE, uklarheter etc.?)
3. [ALLE] Hvordan var oppgaveformuleringene?
4. [ALLE] Var du usikker på hva du skulle gjøre, og hadde det i så fall hjulpet med bedre instruksjoner?
5. [ALLE] Hvordan opplevde du å bruke verktøyene (TAU og SESE) under eksperimentet? (Oppstod det vanskeligheter pga verktøyene? Hvilke konsekvenser hadde det for deg?)
6. [ALLE] Hva synes du generelt om vanskelighetsgraden på oppgavene?
7. [ALLE] Har du vært borti lignende oppgaver før?
8. [ALLE] Jobbet du på eksperimentet forskjellig fra hvordan du jobbet til vanlig?
9. [ALLE] Hvordan var det å sette seg inn i det ferdiglagde programmnet?

10. [ALLE] Var det en bestemt oppgave du syntes var spesielt enkel?
   1. [ALLE] Hvilken oppgave var det?
   2. [ALLE] Hva gjorde at den var enkel?
   3. Hvordan tenkte du da du gjorde endringene for å løse oppgaven?
11. [ALLE] Var det en bestemt oppgave du syntes var spesielt vanskelig?
   1. [ALLE] Hvilken oppgave var det?
   2. [ALLE] Hva gjorde at den var vanskelig?
   3. [ALLE] Hvordan tenkte du da du gjorde endringene for å løse oppgaven?

12. For den oppgaven du synes var vanskeligst [UML]:
   1. Hvordan var det å sette seg inn i UML dokumentasjonen som fulgte med?
   2. I hvilken grad synes du UML dokumentasjonen var nyttig for å løse oppgaven? På hvilken måte?
   3. Hvordan brukte du UML dokumentasjonen i oppgaveløsingen?
   4. Har du brukt UML på denne måten før?
   5. Hva synes du var vanskeligst: Å oppdatere UML diagrammene eller selve Java kodingen?
   6. I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVOR du skulle gjøre endringene?
   7. I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVORDAN du skulle gjøre endringene?

13. For den oppgaven du synes var vanskeligst [IKKE UML]:
   1. Hva synes du var vanskeligst: Å finne ut HVOR du skulle gjøre endringene eller HVORDAN du skulle gjøre endringene?
   2. Hva synes du var vanskeligst: Å finne ut HVOR/HVORDAN du skulle gjøre endringene, eller selve Java kodingen?
   3. I hvilken grad tror du det hadde vært lettere å forstå HVOR du skulle gjøre endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og sekvensdiagrammer)?
   4. I hvilken grad tror du det hadde vært lettere å forstå HVORDAN du skulle gjøre endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og sekvensdiagrammer)?

14. For den oppgaven du synes var enklest [UML]:
   1. I hvilken grad synes du UML dokumentasjonen var nyttig for å løse oppgaven? På hvilken måte?
   2. Hva synes du var vanskeligst: Å finne ut HVOR du skulle gjøre endringene eller HVORDAN du skulle gjøre endringene?
   3. Hva synes du var vanskeligst: Å finne ut HVOR/HVORDAN du skulle gjøre endringene, eller selve Java kodingen?
   4. Hva synes du var vanskeligst: Å oppdatere UML diagrammene eller selve Java kodingen?
   5. I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVOR du skulle gjøre endringene?
   6. I hvilken grad synes du UML dokumentasjonen hjalp deg å forstå HVORDAN du skulle gjøre endringene?
15. For den oppgaven du syntes var lettest [IKKE UML]:
   1. Hva synes du var vanskeligst: Å finne ut HVOR du skulle gjøre endringene eller
      HVORDAN du skulle gjøre endringene?
   2. Hva synes du var vanskeligst: Å finne ut HVOR/HVORDAN du skulle gjøre
      endringene, eller selve Java kodingen?
   3. I hvilken grad tror du det hadde vært lettere å forstå HVOR du skulle gjøre
      endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og
      sekvensdiagrammer)?
   4. I hvilken grad tror du det hadde vært lettere å forstå HVORDAN du skulle gjøre
      endringene dersom du hadde hatt tilgang til UML dokumentasjon (klasse og
      sekvensdiagrammer)?

16. Hvis de som hadde UML ikke benyttet den [UML]:
   1. Hva var grunnen til at du ikke brukte UML dokumentasjon i oppgaveløsningene?
   2. Tror du at hvis du hadde brukt UML dokumentasjon at du hadde fått til en bedre
      løsning?
   3. Hvis du brukte UML litt: hva var det da du så på?
## Appendix C – Experience Questionnaire

1) Erfaring
Dette skjemaet må være ferdig utfylt før du kan begynne på selve eksperimentet. Opplysningene vil utelukkende bli brukt som en del av eksperimentet, og vil bli behandlet konfidensielt.

### Erfaringsskjema

2.1) Arbeidserfaring - Antall år

2.1.1) Hvor mange års arbeidserfaring med programmering har du?

2.1.2) Hvor mange års arbeidserfaring har du totalt?

2.2) Utdannelse - Antall vekttall

2.2.1) Hvor mange vekttall med programmeringsrelatert informatikk har du?

2.2.2) Hvor mange vekttall har du totalt?

2.3) Generell programmeringskompetanse

2.3.1) Hva er din vurdering av hvor dyktig du er som programmerer?

   (1 = Kan svært lite / ingenting, 5 = Ekspert)

   |   |   |   |   |   |
   |   |   |   |   |   |

2.4) Spesifikk programmeringskompetanse – Java

2.4.1) Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?

   (1 = Kan svært lite / ingenting, 5 = Ekspert)

   |   |   |   |   |   |
   |   |   |   |   |   |

2.4.2) Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.5) Spesifikk programmeringskompetanse – C++

2.5.1) Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?

   (1 = Kan svært lite / ingenting, 5 = Ekspert)

   |   |   |   |   |   |
   |   |   |   |   |   |

2.5.2) Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.6) Spesifikk programmeringskompetanse – Simula

2.6.1) Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?

   (1 = Kan svært lite / ingenting, 5 = Ekspert)

   |   |   |   |   |   |
   |   |   |   |   |   |

2.6.2) Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.7) Spesifikk programmeringskompetanse – Small Talk

2.7.1) Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?

   (1 = Kan svært lite / ingenting, 5 = Ekspert)
2.7.2) ***** Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.8) Spesifikk programmeringskompetanse – C
2.8.1) ***** Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?
   (1 = Kan svært lite / ingenting, 5 = Ekspert)

2.8.2) ***** Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.9) Spesifikk programmeringskompetanse – Pascal
2.9.1) ***** Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?
   (1 = Kan svært lite / ingenting, 5 = Ekspert)

2.9.2) ***** Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.10) Spesifikk programmeringskompetanse - Andre språk I
2.10.1) ***** Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?
   (1 = Kan svært lite / ingenting, 5 = Ekspert)

2.10.2) ***** Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

2.11) Spesifikk programmeringskompetanse - Andre språk II
2.11.1) ***** Hva er din vurdering av hvor dyktig du er i dette programmeringsspråket?
   (1 = Kan svært lite / ingenting, 5 = Ekspert)

2.11.2) ***** Estimer omtrentlig hvor mange linjer du har programmert i dette språket:

3) Erfaringsskjema - Hvilke designmetoder/-notasjon kjenner du til, gradert fra 1 til 5
(1 = Kan svært lite, 5 = Ekspert):

3.1) ***** UML

3.2) ***** Verktøyet tau_uml

3.3) ***** Verktøyet Rational Rose

3.4) ***** OMT

3.5) ***** Responsibility-driven design
### 3.6) ***** CRC

| 1 | 2 | 3 | 4 | 5 |

### 3.7) ***** Rollemodellering

| 1 | 2 | 3 | 4 | 5 |

### 3.8) ***** Struktureret analyse og/eller struktureret design

| 1 | 2 | 3 | 4 | 5 |

### 3.9) ***** Data driven design (relasjonsdatabaser el.l.)

| 1 | 2 | 3 | 4 | 5 |

### 3.10) Andre designmetoder/-notasjon du kjenner til

3.10.1) Navn:

| 1 | 2 | 3 | 4 | 5 |

4) Erfaringsskjemaet er nå ferdig utfylt.

Trykk på Lagre-knappen for å avslutte.
Appendix D – Complete Experiment Task Questionnaires

1) Øvelse og dokumentasjon

Introduksjon
I dette eksperimentet skal du implementere endringer på små systemer i Java. Totalt seks oppgaver skal løses. Først skal du gjøre en enkel øvelsesoppgave. Deretter starter selve eksperimentet, som består av fem endringsoppgaver som skal gjennomføres på samme måte som øvelsesoppgaven.

Formål med eksperimentet - konfidenstialitet
Formålet med eksperimentet er ikke å evaluere hvor flink du er til å programmere. Simula Research Laboratory garanterer at navnet ditt vil holdes strengt konfidentielt. Formålet er å evaluere i hvilken grad alternative objektorienterte designstrategier påvirker hvor vanskelig resulterende kode er å endre.

Forberedelser

Gjennomføring av eksperimentet

Selve eksperimentet
Kildekoden som lastes ned inneholder en underkatalog for hvert utviklingsmiljø: Visual J++, Borland JBuilder eller "Other". Du skal bare benytte deg av "Other" og åpne kodefilene i Emacs. Før du laster opp koden med dine implementerte endringer skal du ha gjennomført en kodegjennomgang hvor du forsikrer deg om at besvarelsen produserer (omtrent) samme output som det test-caset som er beskrevet for hver oppgave.

NB! Pass på at du får arbeide uforstyrret med eksperimentet (slå av mobiltelefon). Lykke til.

1.1) Konfidenstialitet
Simula Research Laboratory (SRL) garanterer at individuelle resultater vil bli behandlet strengt konfidentielt. For å sikre gyldighet av resultatene fra eksperimentet bekrefter jeg følgende ved å svare Ja:

1. Jeg har ikke detaljert kjennskap til programvaren som skal endres i dette eksperimentet.
2. Programvaren jeg får tilgang til er SRLs eiendom. Jeg vil ikke kopiere eller lagre kildekoden og binærkode etter at eksperimentet er utført. Spesielt er jeg ansvarlig for å slette eksperimentmaterialet fra lokale filområder når eksperimentet er ferdig.
3. Jeg vil ikke videreformidle opplysninger om dette eksperimentet.

Ja

2) Øvingsoppgave - Del 1

1. Lag en katalog på datamaskinen din som du kaller for Eksperiment.
2. Last så ned en ZIP-filen Training.zip ved å trykke på Last ned-knappen nedenfor og legg den i katalogen.
3) ***** Øvingsoppgave - Del 2
2. Noter tidspunktet i det du begynner å lese oppgaveteksten i feltet nedenfor.
3. Utfor oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
4. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

Last ned

3.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

3.2) Tidsforbruk (i minutter) for å løse oppgaven:
   3.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:
   3.2.2) ***** Tidsforbruk på å implementere endringen:
   3.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):
   3.2.4) ***** Tidsforbruk på å oppdatere designdokumentasjon:

3.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

4) ***** Øvingsoppgave - Del 3
2. Last opp filen Ovelselos.zip ved å trykke på Last opp-knappen nedenfor.

5) Øvingsoppgave - Del 4

Sporreskjema

5.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?
   (1 = Svært eksplorativ, 5 = Svært systematisk)
   1 |__| 2 |__| 3 |__| 4 |__| 5 |__|

5.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
   (1 = Meget dårlig, 5 = Meget bra)
   1 |__| 2 |__| 3 |__| 4 |__| 5 |__|

5.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
   (1 = Meget usikker, 5 = Meget sikker)
   1 |__| 2 |__| 3 |__| 4 |__| 5 |__|
5.4) ***** Hvordan synes du oppgaven var?
(1 = Meget lett, 5 = Meget vanskelig)

1[ ] 2[ ] 3[ ] 4[ ] 5[ ]

5.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

6) ***** Oppgave 1 - Del 1 - MiniBank
   3. Trykk på `Neste`-knappen når dette er gjort.

7) ***** Oppgave 1 - Del 2 - MiniBank
   1. Last ned oppgavebeskrivelsen `Oppgave1.pdf` og legg den i katalogen `Eksperiment`.
   2. Noter tidspunktet i det du begynner å lese oppgaveteksten i feltet nedenfor.
   3. Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
   4. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

7.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

7.2) Tidsforbruk (i minutter) for å løse oppgaven:

7.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

7.2.2) ***** Tidsforbruk på å implementere endringen:

7.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

7.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

8) ***** Oppgave 1 - Del 3 - MiniBank
   2. Last opp filen `Oppg1los.zip` ved å trykke på `Last opp`-knappen nedenfor.

9) Oppgave 1 - Del 4
   MiniBank - Spørreskjema
   9.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?
      (1 = Svært eksplorativ, 5 = Svært systematisk)
Evaluating the Effect of UML on Maintainability

Cand. Scient Thesis

9.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
   (1 = Meget dårlig, 5 = Meget bra)

9.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
   (1 = Meget usikker, 5 = Meget sikker)

9.4) ***** Hvor vanskelig synes du oppgaven var?
   (1 = Meget lett, 5 = Meget vanskelig)

9.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

10) ***** Oppgave 2 - Del 1 - Kaffemaskinen
I de fire siste oppgavene skal du implementere endringer på en virtuell kaffemaskin.
   1. Last ned ZIP-filen Coffee.zip ved å trykke på Last ned-knappen nedenfor og legg den i katalogen Eksperiment.
   3. Trykk på Neste-knappen når dette er gjort.

11) ***** Oppgave 2 - Del 2 Kaffemaskinen
   1. Last ned oppgavebeskrivelsen Oppgave2.pdf og legg den i katalogen Eksperiment.
   2. Noter tidspunktet i det du begynner å lese oppgaveteksten i feltet nedenfor.
   3.Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
   4. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

12) ***** Oppgave 2 - Del 3 - Kaffemaskinen
   1. Zip underkatalogen Coffee, kall Zip-filen for Oppg2los.zip og lagre den i katalogen
13) Oppgave 1 - Del 4
KaffeMaskinen - Spørreskjema

13.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?
   (1 = Svært eksplorativ, 5 = Svært systematisk)
   
1 |__|    2 |__|    3 |__|   4 |__|   5 |__|

13.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
   (1 = Meget dårlig, 5 = Meget bra)
   
1 |__|    2 |__|    3 |__|   4 |__|   5 |__|

13.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
   (1 = Meget usikker, 5 = Meget sikker)
   
1 |__|    2 |__|    3 |__|   4 |__|   5 |__|

13.4) ***** Hvor vanskelig synes du oppgaven var?
   (1 = Meget lett, 5 = Meget vanskelig)
   
1 |__|    2 |__|    3 |__|   4 |__|   5 |__|

13.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

14) ***** Oppgave 3 - Del 1 - Kaffemaskinen
   2. Noter tidspunktet i det du begynner å lese oppgaveområdet i feltet nedenfor.
   3. Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
   4. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

14.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

14.2) Tidsforbruk (i minutter) for å løse oppgaven:

   14.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:
   14.2.2) ***** Tidsforbruk på å implementere endringen:
   14.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

   14.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:
15) ***** Oppgave 3 - Del 2 - Kaffemaskinen
2. Last opp filen Oppg3los.zip ved å trykke på Last opp-knappen nedenfor.

16) Oppgave 1 - Del 4
KaffeMaskinen - Sørgeskjema

16.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?
(1 = Svært eksplorativ, 5 = Svært systematisk)

1 |__| 2 |__| 3 |__| 4 |__| 5 |__|

16.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
(1 = Meget dårlig, 5 = Meget bra)

1 |__| 2 |__| 3 |__| 4 |__| 5 |__|

16.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
(1 = Meget usikker, 5 = Meget sikker)

1 |__| 2 |__| 3 |__| 4 |__| 5 |__|

16.4) ***** Hvor vanskelig synes du oppgaven var?
(1 = Meget lett, 5 = Meget vanskelig)

1 |__| 2 |__| 3 |__| 4 |__| 5 |__|

16.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

17) ***** Oppgave 4 - Del 1 - Kaffemaskinen
2. Noter tidspunktet i det du begynner å lese oppgavebeskrivelsen i feltet nedenfor.
3. Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
4. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

17.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

17.2) Tidsforbruk (i minutter) for å løse oppgaven:

17.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

17.2.2) ***** Tidsforbruk på å implementere endringen:

17.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

17.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:
18) **** Oppgave 4 - Del 2 - Kaffemaskinen
1. Zip underkatalogen coffee, kall Zip-filen for Oppg4los.zip og lagre den i katalogen Eksperiment. (Skriv zip Oppg4los -r Coffee i xterm. (zip <navn_på_filene> -r <navn_på_mappen_som_skal_zippes>))
2. Last opp filen Oppg4los.zip ved å trykke på Last opp-knappen nedenfor.

Last opp

19) Oppgave 1 - Del 4
KaffeMaskinen - Spørreskjema

19.1) **** Hvordan vil du karakterisere din strategi for å løse oppgaven?
(1 = Svært eksplorativ, 5 = Svært systematisk)
1 2 3 4 5

19.2) **** Hva er din subjektive vurdering av kvaliteten på løsningen din?
(1 = Meget dårlig, 5 = Meget bra)
1 2 3 4 5

19.3) **** Hvor sikker er du på at løsningen ikke har alvorlige feil?
(1 = Meget usikker, 5 = Meget sikker)
1 2 3 4 5

19.4) **** Hvor vanskelig synes du oppgaven var?
(1 = Meget lett, 5 = Meget vanskelig)
1 2 3 4 5

19.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

20) **** Oppgave 5 - Del 1 - Kaffemaskinen
1. Last ned oppgavebeskrivelsen Oppgave5.pdf og legg den i katalogen Eksperiment.
2. Noter tidspunktet i det du begynner å lese oppgavebeskrivelsen i feltet nedenfor.
3. Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
4. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

Last ned

20.1) **** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

20.2) Tidsforbruk (i minutter) for å løse oppgaven:

20.2.1) **** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

20.2.2) **** Tidsforbruk på å implementere endringen:

20.2.3) **** Tidsforbruk på å evaluere løsning (kjøre test-case):

- 66 -
20.3) Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

21) Oppgave 5 - Del 2 - Kaffemaskinen
   1. Zip underkatalogen Coffee, kall Zip-filen for Oppg5los.zip og lagre den i katalogen
      Eksperiment: Skriv zip Oppg5los -r Coffee i xterm. (zip <navn_på_fil> -r
      <navn_på_mappen_som_skal_zippes>)
   2. Last opp filen Oppg5los.zip ved å trykke på Last opp-knappen nedenfor.

22) Oppgave 1 - Del 4
    KaffeMaskinen - Spørreskjema

22.1) Hvordan vil du karakterisere din strategi for å løse oppgaven?
      (1 = Svært eksplorativ, 5 = Svært systematisk)
      1 | 2 | 3 | 4 | 5

22.2) Hva er din subjektive vurdering av kvaliteten på løsningen din?
      (1 = Meget dårlig, 5 = Meget bra)
      1 | 2 | 3 | 4 | 5

22.3) Hvor sikker er du på at løsningen ikke har alvorlige feil?
      (1 = Meget usikker, 5 = Meget sikker)
      1 | 2 | 3 | 4 | 5

22.4) Hvor vanskelig synes du oppgaven var?
      (1 = Meget lett, 5 = Meget vanskelig)
      1 | 2 | 3 | 4 | 5

22.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

23) Eksperimentet er ferdig!
    Takk for deltakelsen.
    Trykk på Lagre-knappen for å avslutte eksperimentet.
Evaluating the Effect of UML on **Maintainability**

**Cand. Scient Thesis**

1) Øvelse og dokumentasjon

**Introduksjon**

I dette eksperimentet skal du implementere endringer på små systemer i Java. Totalt seks oppgaver skal løses. Først skal du gjøre en enkel øvelsesoppgave. Deretter starter selve eksperimentet, som består av fem endringsoppgaver som skal gjennomføres på samme måte som øvelsesoppgaven.

**Formål med eksperimentet - konfidensialitet**

Formålet med eksperimentet er *ikke* å evaluere hvor flink du er til å programmere. Simula Research Laboratory garanterer at navnet ditt vil holdes strengt konfidensielt. Formålet er å evaluere i hvilken grad alternative objektorienterte designstrategier påvirker hvor vanskelig resulterende kode er å endre.

**Forberedelser**


**Gjennomføring av eksperimentet**


**Selve eksperimentet**

Kildekoden som lastes ned inneholder en underkatalog for hvert utviklingsmiljø: *Visual J++*, *Borland JBuilder* eller "Other". Du skal bare benytte deg av "Other" og åpne kodefilene i *Emacs*. Før du laster opp koden med dine implementerte endringer skal du ha gjennomført en kodegjennomgang hvor du forsikrer deg om at besvarelsen produserer (omtrent) samme output som det test-caset som er beskrevet for hver oppgave.

NB! Pass på at du får arbeide uforstyrret med eksperimentet (slå av mobiltelefon). Lykke til.

1.1) ***** Konfidensialitet

Simula Research Laboratory (SRL) garanterer at individuelle resultater vil bli behandlet strengt konfidensielt. For å sikre gyldighet av resultatene fra eksperimentet bekrefter jeg følgende ved å svare *Ja*.

Jeg har ikke detaljert kjennskap til programvaren som skal endres i dette

2) ***** Øvingsoppgave - Del 1

5. Lag en katalog på datamaskinen din som du kaller for *Eksperiment*.
7. Skriv *unzip Training.zip* i xterm for å pakke ut filen i samme katalog. Du vil da få opprettet en underkatalog som heter *Training*.
8. Trykk på *Neste*-knappen når dette er gjort.

Last opp
3) ***** Øvingsoppgave - Del 2

7.Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitene (på å forstå oppgaven, på å implementere endringen og på å evaluere løsningen) i feltene nedenfor.
8. Når du er ferdig med å løse oppgaven, noterer du også tidspunktet for dette i feltet nedenfor.

Last ned

3.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

3.2) Tidsforbruk (i minutter) for å løse oppgaven:

3.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

3.2.2) ***** Tidsforbruk på å implementere endringen:

3.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

3.2.4) ***** Tidsforbruk på å oppdatere designdokumentasjon:

3.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

4) ***** Øvingsoppgave - Del 3

4. Last opp filen Ovelselos.zip ved å trykke på Last opp-knappen nedenfor.

Last opp

5) Øvingsoppgave - Del 4

Spørreskjema

5.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?

(1 = Svært eksplorativ, 5 = Svært systematisk)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]

5.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?

(1 = Meget dårlig, 5 = Meget bra)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]

5.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?

(1 = Meget usikker, 5 = Meget sikker)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]

5.4) ***** Hvor vanskelig synes du oppgaven var?

(1 = Meget lett, 5 = Meget vanskelig)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]
5.5) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

6) ***** Oppgave 1 - Del 1 - MiniBank
   6. Trykk på Neste-knappen når dette er gjort.

7) ***** Oppgave 1 - Del 2 - MiniBank
   8.Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen, på å evaluere løsningen og på å oppdatere designdokumentasjonen) i feltene nedenfor.

9) Oppgave 1 - Del 4
   MiniBank - Spørreskjema

9.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

9.2) Tidsforbruk (i minutter) for å løse oppgaven:
   9.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:
   9.2.2) ***** Tidsforbruk på å implementere endringen:
   9.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):
   9.2.4) ***** Tidsforbruk på å oppdatere designdokumentasjonen:

9.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

9.4) ***** Eksporter de oppdaterte UML-diagrammene til eps og lagre dem i underkatalogen MiniBank (dette er beskrevet i skrivet om UML).

9.5) Zip underkatalogen MiniBank (UML-diagrammene og kodeløsning), kall Zip-filen for Oppg1los.zip og lagre den i katalogen Eksperiment: Skriv zip Oppg1los --r MiniBank i xterm. (zip <navn på filen> --r <navn på mappen som skal zippes>)

9.6) Last opp filen Oppg1los.zip ved å trykke på Last opp-knappen nedenfor.
9.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?
(1 = Svært eksplorativ, 5 = Svært systematisk)
1 ◼ 2 ◼ 3 ◼ 4 ◼ 5 ◼

9.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
(1 = Meget dårlig, 5 = Meget bra)
1 ◼ 2 ◼ 3 ◼ 4 ◼ 5 ◼

9.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
(1 = Meget usikker, 5 = Meget sikker)
1 ◼ 2 ◼ 3 ◼ 4 ◼ 5 ◼

9.4) ***** Hvor vanskelig synes du oppgaven var?
(1 = Meget lett, 5 = Meget vanskelig)
1 ◼ 2 ◼ 3 ◼ 4 ◼ 5 ◼

9.5) ***** Når foretok du oppdateringen på designdokumentasjonen?
☐ Til å begynne med
☐ Underveis
☐ Til slutt
☐ Har ikke oppdatert

9.6) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

10) ***** Oppgave 2 - Del 1 - Kaffemaskinen
4. I de fire siste oppgavene skal du implementere endringer på en virtuell kaffemaskin.
5. Last ned ZIP-filen Coffee.zip ved å trykke på Last ned-knappen nedenfor og legg den i katalogen Eksperiment.
7. Trykk på Neste-knappen når dette er gjort.
   Last ned

11) ***** Oppgave 2 – Del 2 Kaffemaskinen
5. Last ned oppgavebeskrivelsen Oppgave2.pdf og legg den i katalogen Eksperiment.
8. Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen, på å evaluere løsningen og på å oppdatere designdokumentasjonen) i feltene nedenfor.
   Last ned

11.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:
11.2) Tidsforbruk (i minutter) for å løse oppgaven:

11.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

11.2.2) ***** Tidsforbruk på å implementere endringen:

11.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

11.2.4) ***** Tidsforbruk på å oppdatere designdokumentasjon:

11.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

12) ***** Oppgave 2 - Del 3 - Kaffemaskinen

3. Eksporter de oppdaterte UML-diagrammene til eps og lagre dem i underkatalogen coffee
(dette er beskrevet i skriven om UML).

4. Zip underkatalogen coffee (UML-diagrammene og kodeløsning), kall Zip-filen for
Oppg2los.zip og lagre den i katalogen Eksperiment: Skriv zip Oppg2los -r coffee i xterm. (zip
<navn_på_filnen> –r <navn_på_mappen_som skal zippes>)

5. Last opp filen Oppg2los.zip ved å trykke på Last opp-knappen nedenfor.

13) Oppgave 1 - Del 4
KaffeMaskinen - Spørreskjema

13.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven? (1 = Svært eksplo- 
    rativ, 5 = Svært systematisk)

13.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din? (1 = Meget dårlig, 5 = Meget bra)

13.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil? (1 = Meget usikker, 5 = Meget sikker)

13.4) ***** Hvor vanskelig synes du oppgaven var? (1 = Meget lett, 5 = Meget vanskelig)

13.5) ***** Når foretok du oppdateringen på designdokumentasjonen?

13.6) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.
14) ***** Oppgave 3 - Del 1 - Kaffemaskinen
7. NB! Husk å oppdatere diagrammene i tau_uvm slik at de samsvarer med løsningen din.
8.Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen, på å evaluere løsningen og på å oppdatere designdokumentasjonen) i feltene nedenfor.

Last ned

14.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

14.2) Tidsforbruk (i minutter) for å løse oppgaven:

14.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

14.2.2) ***** Tidsforbruk på å implementere endringen:

14.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

14.2.4) ***** Tidsforbruk på å oppdatere designdokumentasjon:

14.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

15) ***** Oppgave 3 - Del 2 - Kaffemaskinen
3. Ekспорter de oppdaterte UML-diagrammene til eps og lagre dem i underkatalothen Coffee (dette er beskrevet i skrivet om UML).
5. Last opp filen Oppg3los.zip ved å trykke på Last opp-knappen nedenfor.

Last opp

16) Oppgave 1 - Del 4
KaffeMaskinen - Spørreskjema

16.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven?
(1 = Svært eksplorativ, 5 = Svært systematisk)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]

16.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
(1 = Meget dårlig, 5 = Meget bra)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]

16.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
(1 = Meget usikker, 5 = Meget sikker)

1 [ ] 2 [ ] 3 [ ] 4 [ ] 5 [ ]
16.4) Hvor vanskelig synes du oppgaven var?
   (1 = Meget lett, 5 = Meget vanskelig)
   \[ 1 \square 2 \square 3 \square 4 \square 5 \square \]

16.5) Når foretok du oppdateringen på designdokumentasjonen?

   [ ] Til å begynne med
   [ ] Underveis
   [ ] Til slutt
   [ ] Har ikke oppdatert

16.6) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

<table>
<thead>
<tr>
<th>17) Oppgave 4 - Del 1 - Kaffemaskinen</th>
</tr>
</thead>
<tbody>
<tr>
<td>7. NB! Husk å oppdatere diagrammene i tau.uml slik at de samsvare med løsningen din.</td>
</tr>
<tr>
<td>8.Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen, på å evaluere løsningen og på å oppdatere designdokumentasjonen) i feltene nedenfor.</td>
</tr>
</tbody>
</table>

**Last ned**

17.1) Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

17.2) Tidsforbruk (i minutter) for å løse oppgaven:

17.2.1) Tidsforbruk på å forstå hvordan oppgaven skulle løses:

17.2.2) Tidsforbruk på å implementere endringen:

17.2.3) Tidsforbruk på å evaluere løsning (kjøre test-case):

17.2.4) Tidsforbruk på å oppdatere designdokumentasjon:

17.3) Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

<table>
<thead>
<tr>
<th>18) Oppgave 4 - Del 2 - Kaffemaskinen</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. Ekспорter de oppdaterte UML-diagrammene til eps og lagre dem i underkatalogen Coffee (dette er beskrevet i skrivet om UML).</td>
</tr>
<tr>
<td>5. Last opp filen Oppg4los.zip ved å trykke på Last opp-knappen nedenfor.</td>
</tr>
</tbody>
</table>

Last opp

<table>
<thead>
<tr>
<th>19) Oppgave 1 - Del 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>KaffeMaskinen - Spørreskjema</td>
</tr>
</tbody>
</table>

19.1) Hvordan vil du karakterisere din strategi for å løse oppgaven?
   (1 = Svært eksplorativ, 5 = Svært systematisk)
Evaluating the Effect of UML on **Maintainability**

### 19.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din?
(1 = Meget dårlig, 5 = Meget bra)

### 19.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil?
(1 = Meget usikker, 5 = Meget sikker)

### 19.4) ***** Hvor vanskelig synes du oppgaven var?
(1 = Meget lett, 5 = Meget vanskelig)

### 19.5) ***** Når foretok du oppdateringen på designdokumentasjonen?

- [ ] Til å begynne med
- [ ] Underveis
- [ ] Til slut
- [ ] Har ikke oppdatert

### 19.6) Andre kommentarer, f.eks. uforutsette faktorer som påvirket tidsforbruk.

---

### 20) ***** Oppgave 5 - Del 1 - Kaffemaskinen

7. NB! Husk å oppdatere diagrammene i tau.uml slik at de samsvarer med løsningen din.
8. Utfør oppgaven som er beskrevet i dokumentet og noter samtidig hvor lang tid du bruker på de ulike aktivitetene (på å forstå oppgaven, på å implementere endringen, på å evaluere løsningen og på å oppdatere designinformasjonen) i feltene nedenfor.

#### Last ned

20.1) ***** Noter tidspunktet (tt:mm) for når du startet på oppgaven her:

20.2) Tidsforbruk (i minutter) for å løse oppgaven:

20.2.1) ***** Tidsforbruk på å forstå hvordan oppgaven skulle løses:

20.2.2) ***** Tidsforbruk på å implementere endringen:

20.2.3) ***** Tidsforbruk på å evaluere løsning (kjøre test-case):

20.2.4) ***** Tidsforbruk på å oppdatere designdokumentasjon:

20.3) ***** Noter tidspunktet (tt:mm) for når du avsluttet oppgaven her:

---

### 21) ***** Oppgave 5 - Del 2 - Kaffemaskinen
3. Eksporter de oppdaterte UML-diagrammene til eps og lagre dem i underkatalogen Coffee (dette er beskrevet i skrivet om UML).

4. Zip underkatalogen Coffee (UML-diagrammene og kodeløsning), kall Zip-filen for Oppg5los.zip og lagre den i katalogen Eksperiment: Skriv zip Oppg5los -r Coffee i xterm. (zip <navn på filen> -r <navn på mappen som skal zippes>)

5. Last opp filen Oppg5los.zip ved å trykke på Last opp-knappen nedenfor.

<table>
<thead>
<tr>
<th>22) Oppgave 1 - Del 4</th>
<th>KaffeMaskinen - Spørreskjema</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1) ***** Hvordan vil du karakterisere din strategi for å løse oppgaven? (1 = Svært eksplorativ, 5 = Svært systematisk)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>22.2) ***** Hva er din subjektive vurdering av kvaliteten på løsningen din? (1 = Meget dårlig, 5 = Meget bra)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>22.3) ***** Hvor sikker er du på at løsningen ikke har alvorlige feil? (1 = Meget usikker, 5 = Meget sikker)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>22.4) ***** Hvor vanskelig synes du oppgaven var? (1 = Meget lett, 5 = Meget vanskelig)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>22.5) ***** Når foretok du oppdateringen på designdokumentasjonen?</td>
<td></td>
</tr>
<tr>
<td>☐ Til å begynne med</td>
<td></td>
</tr>
<tr>
<td>☐ Underveis</td>
<td></td>
</tr>
<tr>
<td>☐ Til slutt</td>
<td></td>
</tr>
<tr>
<td>☐ Har ikke oppdatert</td>
<td></td>
</tr>
<tr>
<td>22.6) Andre kommentarer, f.eks. uforutsette faktorer som påvirker tidsforbruk.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23) Spørsmål til pop-up-vinduene</th>
<th>Besvar påstandene nedenfor ved å angi verdi.</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.1) ***** Jeg er vant med å skrive kommentarer når jeg programmerer. (1 = Helt enig med påstanden, 7 = Helt uenig med påstanden)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>23.2) ***** Pop-up-vinduet dukket opp alltfor ofte. (1 = Helt enig med påstanden, 7 = Helt uenig med påstanden)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
23.3) ***** Pop-up-vinduet forstyrret meg i min jobb.  
(1 = Helt enig med påstanden, 7 = Helt uenig med påstanden)  
1 |__| 2 |__| 3 |__| 4 |__| 5 |__| 6 |__| 7 |__|

23.4) ***** Det var nok tid til å skrive kommentarene i pop-up-vinduet.  
(1 = Helt enig med påstanden, 7 = Helt uenig med påstanden)  
1 |__| 2 |__| 3 |__| 4 |__| 5 |__| 6 |__| 7 |__|

23.5) ***** Jeg jobbet annerledes fordi jeg måtte skrive kommentarene i pop-up-vinduet.  
(1 = Helt enig med påstanden, 7 = Helt uenig med påstanden)  
1 |__| 2 |__| 3 |__| 4 |__| 5 |__| 6 |__| 7 |__|

24) Eksperimentet er ferdig!  
Takk for deltakelsen.  
Trykk på `Lagre`-knappen for å avslutte eksperimentet.
Appendix E – Java Code

CoffeeMachine Code

class CoffeeMain
{
/**
 * The main entry point for the application.
 *
 * @param args Array of parameters passed to the application
 * via the command line.
 */
public static void main (String[] args)
{
   CoffeeMachine cm = new CoffeeMachine();
   while (cm.doOneAction())
   {
      //just loop
   }
}
}

class CoffeeMachine
{
   // process input and sends it to the responsible hardware
   private CashBox cashBox;
   private FrontPanel frontPanel;
   private DispenserRegister dispenserReg;
   private ProductRegister productReg;

   // constructor function called when CoffeeMachine is created
   CoffeeMachine()
   {
      cashBox = new CashBox();
      frontPanel = new FrontPanel();
      dispenserReg = new DispenserRegister();
      productReg = new ProductRegister();

      // Load-up the ingredients - normally would
      // obtain externally:
      Ingredient cup = new Ingredient("cup");
      Ingredient coffee = new Ingredient("coffee");
      Ingredient creamer = new Ingredient("creamer");
      Ingredient sugar = new Ingredient("sugar");
      Ingredient water = new Ingredient("water");

      dispenserReg.addDispenser(new Dispenser(cup, 30)); //30 cups
      dispenserReg.addDispenser(new Dispenser(coffee, 10));
      dispenserReg.addDispenser(new Dispenser(creamer, 2));
      // only two shots of cream
      dispenserReg.addDispenser(new Dispenser(sugar, 10));
      dispenserReg.addDispenser(new Dispenser(water, 30));
   }
}
// Load-up the Products - normally would obtain externally:
Product black = new Product
  ("black", 5, new Recipe(cup, coffee, water, null, null));
Product white = new Product
  ("white", 5, new Recipe(cup, coffee, creamer, water, null));
Product sweet = new Product
  ("sweet", 5, new Recipe(cup, coffee, sugar, water, null));
Product whiteSweet = new Product
  ("whiteSweet", 5, new Recipe(cup, coffee, sugar, creamer, water));

productReg.addProduct(black); // product 1
productReg.addProduct(white); // product 2 is coffee with cream
productReg.addProduct(sweet); // product 3 is coffee with sugar
productReg.addProduct(whiteSweet); // product 4 is coffee with cream & sugar

public boolean doOneAction()
{
  // Reads commands and executes them.
  String command;
  Output.print("Menu: I=insert S=select Q=quit");
  command = Input.readString();

  if (!command.equals("Q"))
  {
    if (command.equals("I"))
    {
      int value;
      Output.print("Amount>");
      value = Input.readInt();
      cashBox.deposit(value);
    }
    else if (command.equals("S"))
    {
      int selection;
      Output.print("Select Drink (1 = Black Coffee, 2=Coffee w/Cream, 3=Coffee w/Sugar,
4=Coffee w/Sugar & Cream)>");
      selection = Input.readInt();
      frontPanel.select(selection, cashBox, productReg, dispenserReg);
    }
    return true;
  }
  else return false;
}

class FrontPanel
{
  // instructs the correct product to make the drink
  // instructs CashBox to return change

  public void select(int choiceIndex, CashBox cashBox, ProductRegister productReg,
    DispenserRegister dispenserReg)
class Product
{
    private String myName;
    private Recipe myRecipe;
    private int myPrice;

    Product(String name, int price, Recipe recipe)
    {
    }
}

class CashBox
{
    private int credit = 0;    // knows about money; gives change
    public void deposit(int amount)  // user deposits money
    {
        Output.print("CashBox: Depositing "+ amount);
        credit = credit + amount;
        Output.print("You now have "+ credit + " credits.
");
    }
    // return the currently available credit (change) to the user
    public void returnCoins()
    {
        Output.print("CashBox: Returning "+ credit + \
"\n");
        credit = 0;
    }
    // returns true if user has deposited "amount" or more money
    public boolean haveYou(int amount)
    {
        return (credit >= amount);
    }
    public void deduct(int amount)    // take amount from the credit, return the rest.
    {
        credit = credit - amount;
        returnCoins();
    }
}

// class Product:
// Abstraction of the drink.
// Responsible for knowing its name and recipe (for now).
class Product
{
    private String myName;
    private Recipe myRecipe;
    private int myPrice;

    Product(String name, int price, Recipe recipe)
    {
    }
}
myName = name;
myPrice = price;
myRecipe = recipe;
}
public String name()
{
    return myName;
}
public Recipe recipe()
{
    return myRecipe;
}
public int price()
{
    return myPrice;
}
public void makeDrink(DispenserRegister dispenserReg)
{
    myRecipe.makeDrink(dispenserReg);
}

import java.util.Vector;
// class ProductRegister:
// Abstraction of the thing that holds all the products.
// Knows what products are available.
class ProductRegister
{
    private Vector products = new Vector();
    public Product productFromIndex(int index)
    {
        return (Product) products.elementAt(index-1);
    }
    public void addProduct(Product p)
    {
        products.addElement(p);
    }
}

// knows its ingredient, knows how to dispense a shot of the ingredient
class Dispenser
{
    private Ingredient ingredient;
    private int shotsLeft;

    // constructor function called when creating a new dispenser:
    Dispenser(Ingredient ingr, int initialamount)
    {
        ingredient = ingr;
        shotsLeft = initialamount;
    }

    public boolean dispense()
    {

// dispense one portion of the ingredient
if (shotsLeft == 0)
{
    Output.print("\tDispenser: out of " + ingredient.name() + "\n");
    return false;
}
else
{
    Output.print("\tDispensing " + ingredient.name() + "\n");
    shotsLeft = shotsLeft - 1;
    return true;
}

// returns the ingredient contained in this dispenser
public boolean contains(Ingredient i)
{
    return i.equals(ingredient);
}

import java.util.Vector;
// contains all the dispensers, knows which dispenser contains a given ingredient
class DispenserRegister
{
    private Vector dispensers = new Vector();
    // add dispenser to the vector
    public void addDispenser(Dispenser dispenser)
    {
        dispensers.addElement(dispenser);
    }
    // find the dispenser for a given ingredient
    public Dispenser getDispenserOf(Ingredient ingr)
    {
        int index = 0;
        while (index < dispensers.size())
        {
            Dispenser d = (Dispenser) dispensers.elementAt(index);
            if (d.contains(ingr)) return d;
            index = index + 1;
        }
        return null;
    }
}

import java.util.Vector;
// Abstraction of a recipe.
// Tells the dispensers to dispense ingredients in sequence.
class Recipe
{
    private Vector ingredients = new Vector();
Recipe(Ingredient i1, Ingredient i2, Ingredient i3, Ingredient i4, Ingredient i5)
{
    addIngredient(i1);    addIngredient(i2);    addIngredient(i3);    addIngredient(i4);    addIngredient(i5);}
private void addIngredient(Ingredient ingr)
{    if(ingr != null) ingredients.addElement(ingr);}
public void makeDrink(DispenserRegister dispenserReg)
{    int i = 0;    while (i<ingredients.size())    {        Ingredient nextIngr = (Ingredient) ingredients.elementAt(i);        dispenserReg.getDispenserOf(nextIngr).dispense();        i = i + 1;    }}
}
// ingredient is just a string with the ingredient name
class Ingredient
{    private String myName;    // constructor function called when object is created, f.ex.,
    // Ingredient i = new Ingredient("sugar");
    Ingredient(String name)    {        myName = name;    }    public String name()    {        return myName;    }}
// Example use of the Input class:
// String s;
// s = Input.readString();
//
// int count;
// Output.print("Please enter number of values");
// count = Input.readInt();
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

- 83 -
import java.lang.Integer;

class Input
{
    private static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    public Input() {}
    // read a string from the standard input
    public static String readString()
    {
        String s;
        try
        {
            s = in.readLine();
            s = s.toUpperCase();
            return s;
        }
        catch (IOException e)
        {
            return "";
        }
    }
    // read in integer from the standard input
    public static int readInt()
    {
        try
        {
            String s = in.readLine();
            if (s.length() > 0)
            {
                return Integer.parseInt(s);
            }
            else
            {
                return -1;
            }
        }
        catch (IOException e)
        {
            return -1;
        }
    }

    // Example use of the Output class:
    // Output.print("Hello World");
    //
    // int count = 5;
    // Output.print("Number of values = " + i)
    class Output
    {
        public static void print(String s)
        {
            System.out.println(s);
        }
    }
}
MiniBank Code

public class MiniBankMain
{
    /* The main entry point for the application.
    * @param args Array of parameters passed to the application
    * via the command line.
    */
    public static void main (String[] args)
    {
        MiniBank miniBank;
        miniBank = new MiniBank();
        miniBank.runBank();
        // the program terminates.
    }
}

class MiniBank
{
    private AccountRegister TheAccountReg;
    private CashHandler TheCashHandler;

    // public methods:
    public void runBank()
    {
        boolean quit = false;
        // initialize IO
        TheCashHandler = new CashHandler();
        TheCashHandler.initCash(1000);
        TheAccountReg = new AccountRegister();
        TheAccountReg.initRegister();

        while (!quit)
        {
            quit = handleMenuChoice();
        }
    }

    // private methods:
    private String getMenuChoice()
    {
        // show the menu and return the user's menu selection:
        Output.print("Menu: N = New account W=Withdraw. D=Deposit.
                   S=Statement Q=Quit");
        return Input.readString();
    }
}
private boolean handleMenuChoice()
{
    boolean quit = false;
    String choice;

    choice = getMenuChoice();
    if (choice.equals("N"))
    {
        newAccount();
    }
    else if (choice.equals("W"))
    {
        withdraw();
    }
    else if (choice.equals("D"))
    {
        deposit();
    }
    else if (choice.equals("S"))
    {
        statement();
    }
    else if (choice.equals("Q"))
    {
        quit = true;
    }
    else
    {
        Output.print("Wrong menu choice. Please try again.");
    }
    return quit;
}

private Account authenticate()
{
    // read account number and pin-code from the user, and authenticate.
    // If authentication ok, return the Account, otherwise, return null.

    String accountnum;
    int pincode;
    Account acc;

    // Get the account number from the user
    Output.print("Enter your account number: ");
    // Read the account number from the keyboard
    accountnum = Input.readString();
    acc = TheAccountReg.findAccount(accountnum);
    if (acc != null)
    {
        Output.print("Enter your pin code: ");
    }
pincode = Input.readInt();
if (!acc.isValidPinCode(pincode))
{
    Output.print("Invalid PIN code");
    acc = null;
}
else
{
    Output.print("Account does not exist");
}
return acc;

private void newAccount()
{
    // read account number and pin-code from the user, and authenticate.
    // If authentication ok, return the Account, otherwise, return null.
    String accountnum;
    int pincode;
    Account acc;

    Output.print("Enter a new account number: "); // Print a message.
    accountnum = Input.readString(); // Read the first account from input
    acc = TheAccountReg.findAccount(accountnum);
    if (acc != null)
    {
        Output.print("Sorry, account number already exists: ");
    }
    else
    {
        Output.print("Please enter a personal pin code: ");
        pincode = Input.readInt();
        TheAccountReg.addAccount(accountnum, pincode);
        Output.print("New account has been created.");
    }
}

private void withdraw()
{
    Account acc;
    acc = authenticate();
    if (acc != null) {
        int amount;
        Output.print("Enter amount: ");
        amount = Input.readInt();
        // must check if there is enough money in account and in dispenser
        if ((acc.getAmount()>= amount))
        {
            if (TheCashHandler.getAmount()>=amount)
            {
                TheCashHandler.dispenseCash(amount);
                acc.withdraw(amount);
            }
            else
            {
            }
        }
    }
}
private void deposit()
{
    Account acc;
    acc = authenticate();
    if (acc != null) {
        int amount=0;
        Output.print("Insert money: ");
        amount = TheCashHandler.insertCash();
        if (amount > 0)
        {
            acc.deposit(amount);
        }
    }
}

private void statement()
{
    Account acc;
    acc = authenticate();
    if (acc != null) {
        Output.print("Sorry, transaction statement not implemented yet");
        // NOT IMPLEMENTED
    }
}

class Account
{
    private String AccountNumber;
    private int PinCode;
    private int Amount;

    public void createAccount(String accnum, int pin)
    {
        this.AccountNumber = accnum;
        this.PinCode = pin;
        this.Amount = 0; // start with no money
    }

    public boolean hasAccountNumber(String accnum)
    {
        // returnerer true hvis denne kontoen har
        // samme kontonummer som accnum:

        Output.print("Sorry, insufficient cash in the machine!");
    }

    else
    {
        Output.print("Insufficient funds in the account!");
    }
}
return (this.AccountNumber.equals(accnum));
}

public String getAccountNumber()
{
    return this.AccountNumber;
}

public int getAmount()
{
    return this.Amount;
}

private int getPinCode()
{
    // for future use: could add encryption
    return this.PinCode;
}

public boolean isValidPinCode(int pin)
{
    return (this.PinCode == pin);
}

public boolean withdraw(int outamount)
{
    // this function is called when money is withdrawn from the account
    boolean transactionOK = false;
    if (this.getAmount() >= outamount)
    {
        this.Amount = this.Amount - outamount;
        transactionOK = true;
    }
    else
    {
        Output.print("Not sufficient funds in account");
    }
    return transactionOK;
}

public boolean deposit(int inamount)
{
    // this function is called when money is deposited into the account
    boolean transactionOK = false;
    if (inamount > 0) 
    
    
    
    if (inamount > 0) {
        this.Amount = this.Amount + inamount;
        transactionOK = true;
    }
    else
    {
        // illegal!!
        Output.print("Deposits must be positive!");
    }

}
import java.util.Vector;
public class AccountRegister {
    private Vector allAccounts;

    public void initRegister() {
        allAccounts = new Vector();
    }

    public Account findAccount(String accNum) {
        int i = 0;
        while (i<allAccounts.size()) {
            Account a;
            a = (Account) allAccounts.elementAt(i);
            if (a.hasAccountNumber(accNum)) {
                return a;
            }
            i = i + 1;
        }
        // Didn't find account. Return "nothing".
        return null;
    }

    public Account addAccount(String accNum, int pincode) {
        Account a;
        a = new Account();
        a.createAccount(accNum, pincode);
        this.allAccounts.addElement(a);
        return a;
    }
}

class CashHandler {
    // knows how much money is available in the MiniBank
    // gives the cash to the user
    private int TotalAmount;

    public void initCash(int initialamount) {
        this.TotalAmount = initialamount;
    }

    public int getAmount() {
        return this.TotalAmount;
    }

    private boolean dispense(int amount) {
        // - 90 -
Evaluating the Effect of UML on Maintainability

Cand. Scient Thesis

Output.print("Dispensing " + amount);
return true; // no hardware errors

public boolean dispenseCash(int amount)
{
    boolean transactionOK = false;
    if (amount <= TotalAmount)
    {
        // this line should
        if (dispense(amount)) {
            this.TotalAmount = TotalAmount - amount;
            transactionOK = true;
        }
        else
        {
            Output.print("Sorry, Cash Dispenser is out of order!");
        }
    }
    else
    {
        Output.print("Sorry, not enough cash in the cash dispenser!");
    }
    return transactionOK;
}

private int insert()
{
    // this would be different in a "real" machine
    return Input.readInt();
    // assume no hardware errors in debug version
}

public int insertCash()
{
    int amountinserted = insert();
    // this line should
    if (amountinserted > 0) {
        this.TotalAmount = TotalAmount + amountinserted;
    }
    else
    {
        amountinserted = 0;
        Output.print("Sorry, Cash Insert is out of order!");
    }
    return amountinserted;
}

// Example use of the Input class:
// String s;
// s = Input.readString();
// int count;
// Output.print("Please enter number of values");
// count = Input.readInt();

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.lang.Integer;

class Input {
    private static BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    public Input() {}
    public static String readString() {
        String s;
        try {
            s = in.readLine();
            s = s.toUpperCase();
            return s;
        } catch (IOException e) {
            return "";
        }
    }

    public static int readInt() {
        try {
            String s = in.readLine();
            if (s.length() > 0) {
                return Integer.parseInt(s);
            } else {
                return -1;
            }
        } catch (IOException e) {
            return -1;
        }
    }
}

class Output {
    public static void print(String s) {
        System.out.println(s);
    }
}

// Example use of the Output class: Output.print("Hello World"); int count = 5;
// Output.print("Number of values = " + i)
Appendix F – UML Diagrams

The UML Diagrams for MiniBank (ATM)

Diagram 1: MiniBank Class Diagram
Diagram 2: NewAccount, creating a new account in the ATM

- A new account object is created every time a new account is made. We could have used collections, but we choose to separate the objects.
- The way this is made clear that what really happens.
Diagram 3: Deposit, inserting money to the user's account in ATM
Diagram 4: Withdraw, the user is withdrawing money from the account.
Diagram 5: Statement: this function is not complete. Giving an error message to the user.
Diagram 6: Quit, the user logs out and the scenario is terminated
Diagram 7: CoffeeMachine class diagram
Diagram 8: CoffeeMachine, this overview shows the dataflow of the three functions Insert, Select and Quit. This part shows Select.
Diagram 9: CoffeeMachine, this overview shows the dataflow of the three functions Insert, Select and Quit. This part shows Insert and Quit.
Diagram 10: CoffeeMachine, the user have chosen Black Coffee, this part of the Sequence diagram shows the choosing process.
Diagram 11: CoffeeMachine, rest of the Black Coffee Sequence diagram, this part show the dispensing of the product.
Diagram 12: CoffeeMachine, the user have chosen Coffee with cream, this part of the Sequence diagram shows the choosing process.
Diagram 13: Coffee/Machine, rest of the Sequence diagram Coffee with Cream, this part shows the dispensing of the product
Diagram 14: CoffeeMachine, the user has chosen Coffee with Sugar. This part of the Sequence diagram shows the choosing process.
Diagram 15: CoffeeMachine, rest of the Sequence diagram for Coffee with Sugar, this part show the dispensing of the product
Diagram 16: CoffeeMachine, the user has chosen Coffee with Cream and Sugar. This part of the Sequence diagram shows the choosing process.
Diagram 17: CoffeeMachine, rest of the Sequence diagram for Coffee with Cream and Sugar, this part show the dispensing of the product, left out the Print command in this view, but in the initial diagram it is included.
Appendix G – The Experiments Change Tasks

Change Task Descriptions
(Some details are omitted. Translated from Norwegian)

TRAINING TASK
Complete the code you just downloaded so that it can read an arbitrary number of lines of text from INPUT and stores each string in a Vector. When the user presses <CR>, the program should write the number of lines of text, and thereafter print out the text in the reverse order (that is, the last string should be printed first).

Test case:
Enter a string. Finish with <CR>
abc
Enter next string. Finish with <CR>
def
Enter next string. Finish with <CR>
ghi
Enter next string. Finish with <CR>

You entered 3 strings.
The strings in reverse order are:
ghi
def
abc
CHANGE TASK 1
The code you just downloaded contains a simple automated teller machine (ATM). At present, the ATM has the following functionality:

- **New account**: … (detailed description omitted in this report)
- **Withdraw**: … (detailed description omitted in this report)
- **Deposit**: … (detailed description omitted in this report)

Add the following functionality to the ATM:

- **Account Statement**: Gives an account statement for a customer (menu choice = “Statement”). For every withdrawal a given customer has made, the statement should contain a line "Withdrew <amount>". Similarly, for every deposit the statement should contain a line "Deposited <amount>". Then the current account balance is printed. For details, refer to the following test case.

**Test case:**

```
Menu: N = New account W=Withdraw. D=Deposit. S=Statement Q=Quit
N
Enter a new account number:
per hansen
Please enter a personal pin code:
1234
New account has been created.
Menu: N = New account W=Withdraw. D=Deposit. S=Statement Q=Quit
D
Enter your account number:
per hansen
Enter your pin code:
1234
Insert money:
200
Menu: N = New account W=Withdraw. D=Deposit. S=Statement Q=Quit
W
Enter your account number:
per hansen
Enter your pin code:
1234
Enter amount:
100
Dispensing 100
Menu: N = New account W=Withdraw. D=Deposit. S=Statement Q=Quit
W
Enter your account number:
per hansen
Enter your pin code:
1234
Enter amount:
50
Dispensing 50
Menu: N = New account W=Withdraw. D=Deposit. S=Statement Q=Quit
S
Enter your account number:
per hansen
Enter your pin code:
1234
Withdraw 100
Withdraw 50
----------
Account balance 50
Menu: N = New account W=Withdraw. D=Deposit. S=Statement Q=Quit
```
CHANGE TASK 2
In this task, you shall extend the coffee machine with a "return button" functionality that returns the deposited funds. The menu choice is called "Return".

Test Case:

Menu: I=insert S=select R=return Q=quit
I
Amount>
4
CashBox: Depositing 4
You now have 4 credits.

Menu: I=insert S=select R=return Q=quit
R
CashBox: Returning 4

CHANGE TASK 3
In this task, you shall extend the machine to make bouillon. Bouillon costs more than coffee. While coffee costs 5 credits, bouillon costs 6 credits.

Test Case:

Menu: I=insert S=select R=return Q=quit
I
Amount>
6
CashBox: Depositing 6
You now have 6 credits.

Menu: I=insert S=select R=return Q=quit
S
Select Drink (1 = Black Coffee, 2=Coffee w/Cream, 3=Coffee w/Sugar, 4=Coffee w/Sugar & Cream, 5=Bouillon)>
5
Dispensing cup
Dispensing bouillon
Dispensing water
CashBox: Returning 0

Menu: I=insert S=select R=return Q=quit
CHANGE TASK 4

Unfortunately, there is a quite serious problem with the coffee machine at present. If the user chooses for example "coffee with cream", and the cream dispenser is empty, the machine gives a small error message, after which it dispenses black coffee (without cream). If the machine does not contain any more cups, the machine dispenses the drink right into the drain… The user will of course get quite irritated over having to pay for this!

The simplest solution to this problem is that the user receives a message if the machine is out of a required ingredient of the selected drink. Then, the user is given the option to choose another drink. The following test case illustrates what should happen when the machine runs out of cream:

**Test Case:**

Menu: I=insert S=select R=Return Q=quit

I  
Amount>

CashBox: Depositing 5
You now have 5 credits.

Menu: I=insert S=select R=Return Q=quit

S  
Select Drink (1 = Black Coffee, 2=Coffee w/Cream, 3=Coffee w/Sugar, 4=Coffee w/Sugar & Cream, 5=Bouillon)>

Dispensing cup
Dispensing coffee
Dispensing water
Dispensing cream <after this the machine is out of cream>
CashBox: Returning 0

Menu: I=insert S=select R=Return Q=quit

I  
Amount>

CashBox: Depositing 5
You now have 5 credits.

Menu: I=insert S=select R=Return Q=quit

S  
Select Drink (1 = Black Coffee, 2=Coffee w/Cream, 3=Coffee w/Sugar, 4=Coffee w/Sugar & Cream, 5=Bouillon)>

Dispensing cup
Dispensing coffee
Dispensing water
Dispensing cream <after this the machine is out of cream>
CashBox: Returning 0

Menu: I=insert S=select R=Return Q=quit
CHANGE TASK 5

You are going to make a new menu choice “Make your own drink”, which allows the customer to choose among any combination of available ingredients to make a custom drink (see test-case). Note! There is no checking on whether the combination of ingredients “makes sense”. However, if the machine is (or becomes) empty of a given ingredient, the customer should receive an error message and can then choose an alternative ingredient. Each shot of an ingredient costs 2 credits. If the customer has put on insufficient amounts of money for the chosen set of ingredients, the customer receives the message “Insufficient funds” and thereafter the menu choice “Menu: I=insert S=select R=Return Q=quit”.

Test Case:
Menu: I=insert S=select R=Return Q=quit
1
Amount
10
CashBox: Depositing 10
You now have 10 credits.

Menu: I=insert S=select R=Return Q=quit
S
Select Drink (1 = Black Coffee, 2=Coffee w/Cream, 3=Coffee w/Sugar, 4=Coffee w/Sugar & Cream 5= Bouillon, 6=Make your own drink):>
6
Select Ingredient (1=Cup, 2=Coffee, 3=Sugar, 4=Water, 5=Cream, 6=Bouillon, 0=Make Drink)
1
You have selected cup
This drink costs 2 credits
Select Ingredient (1=Cup, 2=Coffee, 3=Sugar, 4=Water, 5=Cream, 6=Bouillon, 0=Make Drink)
2
You have selected cup, coffee
This drink costs 4 credits
Select Ingredient (1=Cup, 2=Coffee, 3=Sugar, 4=Water, 5=Cream, 6=Bouillon, 0=Make Drink)
2
You have selected cup, coffee, coffee
This drink costs 6 credits
Select Ingredient (1=Cup, 2=Coffee, 3=Sugar, 4=Water, 5=Cream, 6=Bouillon, 0=Make Drink)
4
You have selected cup, coffee, coffee, water
This drink costs 8 credits
Select Ingredient (1=Cup, 2=Coffee, 3=Sugar, 4=Water, 5=Cream, 6=Bouillon, 0=Make Drink)
5
Sorry, no more cream!
You have selected cup, coffee, coffee, water
This drink costs 8 credits
Select Ingredient (1=Cup, 2=Coffee, 3=Sugar, 4=Water, 5=Cream, 6=Bouillon, 0=Make Drink)
0
Dispensing cup
Dispensing coffee
Dispensing coffee
Dispensing water
CashBox: Returning 2

Menu: I=insert S=select R=Return Q=quit
Qualitative Data Collection 35

5.6 Results 37
MiniBank – UML diagrams

Class Diagram
NewAccount

```
minibank_user

Menu N=New Account W=Withdraw D=Deposit S=Statement Q=Quit

selects N

newAccount

print("Enter a new account number")

Enter a new account number

findAccount(accountNum)

hasAccountNumber(accountNum)

createAccount(accountNum, pinCode)

New account has been created

return quit

return null
```

a Bank

Output

Input

TheAccountRegister

Account

Account

A account object is created every time a new account is made. We could have used collections, but we choose to separate the objects this way there is more clear what really happens.
Deposit
account object is created every time a new account is made. we could have used collections, but we choose to separate the objects this way; it is more clear what really happens.
continue
Coffee With cream

User

continue

CoffeeMachine

:Input

Menu:
1=insert
S=select
Q=quit

Coffee Drink selection:
1=Black Coffee
2=Coffee w/Cream
3=Coffee w/Sugar
4=Coffee w/Cream & Sugar

User Choose 2

selection=readInt

return selection=2

select(2, cashBox, productReg, dispenserReg)

productFromIndex

(makeDrink(dispenserReg))

getIt

makeDrink(dispenserReg)
continue

User
cmp: CoffeeMachine

print("Menu
I=insert
S=select
Q=quit")

User selects S
command = readString
return command = S

print("Select Drink")
Select Drink
1=Black Coffee
2=Coffee w/Cream
3=Coffee w/Sugar
4=Coffee w/Cream&Sugar

User chooses?
selection = readInt
return selection = 3
select(3, cashBox, productReg, dispenseReg)

productFromIndex
(3)
return black... haveYou(order)
return true
makeDrink(dispenseReg)
makeDrink(dispenseReg)
getDisp
Coffee with sugar and cream