The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation

Erik Arisholm[§], Lionel C. Briand[‡§], Siw Elisabeth Hove[§], Yvan Labiche[‡]

[‡] Carleton University
Department of Systems and Computer Engineering
Software Quality Engineering Laboratory
1125 Colonel By Drive
Ottawa, ON K1S5B6, Canada
{briand, labiche}@sce.carleton.ca

[§] Simula Research Laboratory
Department of Software Engineering
Martin Linges v 17, Fornebu
P.O.Box 134,
1325 Lysaker, Norway
{erika,siweh}@simula.no

ABSTRACT

The Unified Modeling Language (UML) is becoming the de-facto standard for software analysis and design modeling. However, there is still a significant resistance to model-driven development in many software organizations as it is perceived to be expensive and not necessarily cost-effective. It is therefore important to investigate the benefits obtained through modeling. As a first step in this direction, this paper reports on controlled experiments, spanning across two locations, which investigate the impact of UML documentation on software maintenance. Results show that, for complex tasks and past a certain learning curve, the availability of UML documentation may result in significant improvements of the functional correctness of changes as well as their design quality. On the other hand, there does not seem to be any resulting time saving. For simpler tasks, the time needed to update the UML documentation may be substantial compared with the potential benefits, thus motivating the need for UML tools with better support for software maintenance.

1 INTRODUCTION

Software maintenance is often performed by individuals who were not involved in the original design of the system being changed. This is why documenting software specifications and designs often has been advocated as a necessity to help software engineers remain in intellectual control while changing complex systems [7, 11]. Indeed, it is expected that many aspects of a software system need to be understood in order to

properly change it, including its functionality, architecture, and a myriad of design details.

However, documentation entails a significant cost and it must be maintained along with the software system it describes. The question that arises is then what should be the content and level of detail required to help software maintenance [9]? The proponents of agile methods usually advocate to keep documentation to a minimum and to focus on test cases as a source of system requirements [6]. On the other hand, model-driven development views software development as a series of modeling steps, each of them refining the models of the previous step [18]. The transition from analysis, to high-level design, low-level design, and then code is supported by tools facilitating and automating parts of the work. Modeling is seen as a way to better handle the growing complexity of software development by helping engineers work at higher levels of abstraction. Model-driven development is supported by the Unified Modeling Language (UML) [7], an evolving standard which is now widespread across the software industry. Despite a growing popularity, there is little reported evaluation of the use of UML-based development methods [1], and many perceive the documentation of analysis and design models in UML to be a wasteful activity [6]. Such practices are therefore viewed as difficult to apply in development projects where resources and time are tight.

It is then important, if not crucial, to investigate whether the use of UML documentation can make a practically significant difference that would justify the costs. This is particularly true in the context of software maintenance which consumes most of software development resources [28] and entails the comprehension of large, complex systems under constant change.

This paper attempts to evaluate the impact of using UML modeling on the correctness and effort of performing changes. This is done through two controlled experiments taking place in two distinct geographical locations. Both involve students with substantial training in object-oriented programming and UML modeling but coming from two different education systems. An additional objective is to identify reasons for varying results and therefore identify plausible and necessary conditions for UML to be effective.

Our decision to answer the above research question with controlled experiments stems from the many confounding and uncontrollable factors that could blur the results in an industrial context. It is usually impossible to control for factors such as ability and learning/fatigue effects, and select specific tasks to assign to individuals in a real project setting. As a result, the threats to internal validity are such that it is difficult to establish a causal relationship between independent (e.g., UML) and dependent variables (e.g., time, correctness). Basili *et al.* [5] state that "Controlled experiments can generate stronger statistical confidence in the conclusions." and Judd *et al.* [16] write that " ... we can confidently infer causality from the relationship between two variables only if people have been randomly assigned to the levels of the independent variables."

However, controlled experiments are a compromise as they can only span limited time and necessarily involve smaller tasks and artifacts. Again, this is a well known threat: "Unfortunately, since controlled experiments are expensive and difficult to control if the project is too large, the projects studied tend to be small." [4] It therefore raises questions about the extent to which their results can be generalized to realistic tasks, artifacts, and project settings (external validity). From a pragmatic standpoint, both types of studies are needed to obtain a credible body of evidence, but during the earlier stages of an investigation, controlled experiments enable the investigators to better understand the issues at stake and the factors to be accounted for. Furthermore, one can analyze whether the results obtained on smaller artifacts and tasks can at least be considered encouraging and justify further evaluation in more realistic settings.

The rest of the article is structured as follows. Related work is discussed in Section 2. Section 3 reports on the planning of the two controlled experiments and results are presented in Section 4. Section 5 analyses threats to validity and conclusions are drawn in Section 6.

2 RELATED WORKS

In the following we discuss a number of studies that have investigated the impact of program documentation, specific ways of using UML or compared UML to other notations in the context of program comprehension and maintenance.

In [36], an experiment was conducted to assess the qualitative efficacy of UML diagrams in aiding program understanding. The subjects, who the authors rated as UML experts, had to analyze a series of UML diagrams and answer a detailed questionnaire concerning a hypothetical software system. Results from the experiment suggest that the UML's efficacy in support of program understanding is limited by (1) unclear specifications of syntax and semantics in some of UML's more advanced features, (2) spatial layout problems, e.g., large diagrams are not easy to read, and (3) the UML's insufficient support for representing the domain knowledge required to understand a program.

A complementary work, that also investigated the impact of UML documents' content, investigated whether making UML models more precise using the Object Constraint Language (OCL) [37]—which is part of the UML standard [23]—helped with defect detection, comprehension, and maintenance [10]. The results showed that, once past an initial learning curve, significant benefits can be obtained by using OCL in combination with UML analysis diagrams to form a precise UML analysis model. However, this result was conditioned on providing substantial, thorough training to the experiment participants.

Other studies have compared UML to other notations in specific contexts: For example, the experiment reported in [32] compared the comprehension of subjects when provided with two different types of modeling notations: UML models or OPM (Object-Process Methodology) models. The results suggest that OPM is better than UML in modeling the dynamics aspect of Web applications for relatively untrained users. Another experiment, reported in [26], also evaluated the comprehensibility of the dynamic models in two standard languages, UML versus OML (OPEN Modeling Language). The results

suggested that the specification of the dynamic behavior using OML is faster to comprehend and easier to interpret than using the UML language.

Other works have studied the comprehensibility of alternative UML diagramming notations, different layout heuristics and syntactic variations of UML-models. For example, an experiment on the comprehension of sequence versus collaboration diagrams was presented in [19], and showed no significant differences in the understandability of the two alternative notations. An experiment reported in [20] showed that the use of stereotypes had a positive effect on model understandability. Experiments reported in [29, 30] investigated the impact of semantically identical but syntactically or stylistically different variations of UML class and collaboration diagrams. Results showed that which variation was "best" depended on the task for which it was used.

A controlled experiment reported in [2] investigated how access to *textual* system documentation (the requirements specification, design document, test report, and user manual) helped when performing maintenance tasks. The results indicated that having documentation available during system maintenance reduces the time needed to understand how to perform maintenance tasks by approximately 20 percent. The results also suggested that there is an interaction between the maintainer's skill (as indicated by a pre-test score) and the potential benefits of the system documentation: the most skilled maintainers benefited the most from the documentation.

In [3], the authors investigated the impact of the design style of an Object-Oriented (OO) system on its understandability. The two design styles of interest were a delegated control style, often advocated by modern OO development methodologies, and a centralized control style, often regarded as reminiscent of a procedural solution. The results suggested that the delegated control style was very difficult to understand for novices. Only senior developers benefited from a delegated control style. One possible explanation of the results reported in [3] is discussed in [34]: *Delocalized plans* need to be explicitly documented in higher level representations of the code to aid in program understanding and maintenance. The authors question whether it is even reasonable to look at the details of program code in order to understand a (delocalized) program [34]:

"why should programmers look at program text? That is, we do not need to look at the compiled version of a program because the details are overwhelming."

One investigation evaluating whether using UML is cost-effective in a realistic context, for a large project, has been recently conducted under the form of a qualitative case study [2]. The participants in the case study acknowledged that, despite some difficulties (e.g., need for adequate training), there are clear benefits from using UML (e.g., traceability from functional requirements to code).

However, to the best of the authors' knowledge, none of the existing works investigated by means of a controlled experiment the fundamental question of whether UML documentation yields practical benefits when changing systems. This is however crucial if we want to see widespread adoption of model-driven development in industry and convince software managers and engineers that UML modeling is really worth the effort. Furthermore, most of the experiments above did not involve actual changes to UML models, and none of them involved changes to code. Their experimental tasks mostly involved responding to comprehension questions. The current paper investigates the impact of UML in the specific context of software maintenance tasks while performing actual changes to two systems. The current experiment complements the results of qualitative case studies such as those reported in [2].

3 EXPERIMENT PLANNING

In this section we report two distinct controlled experiments, following the template proposed in [38]. Though the research question they pose is identical, they differ in terms of their design, their size, and their focus. The first experiment took place in Oslo, Norway and the second one in Ottawa, Canada. We will refer to them as the Oslo and Ottawa experiments, respectively. The Oslo experiment involved less participants and because it was the first one, it was more exploratory and therefore relied more heavily on subject interviews and qualitative analysis [33]. The Ottawa experiment involved a larger number of participants and was designed to make standard statistical analyses possible. Further details on the differences between the two experiments are reported below.

3.1 Experiment Definition

The UML documentation used here corresponds to what you would typically expect at the end of design, in terms of content and level of detail [11], as further described in Section 3.5.1. We first want to assess whether the provided UML documentation helps shorten the effort required to change the source code. In other words, we are interested in whether UML documents can help reduce the costs related to code changes and, in order to perform such an analysis, we must measure the time required to complete the maintenance tasks of our experiment. Two different measures of time are considered as we may choose to include or not the time taken for modifying UML models. Though it is only fair to account for the time required to modify models to assess whether time is saved, the time spent on modifying the models is probably strongly dependent on the modeling tool used and the subject's training in that particular tool. This is highly context-dependent and we therefore wanted to distinguish the time people spent understanding and modifying the code (with the help of UML diagrams) from the time spent on modifying the UML diagrams. Both measures of time are expected to provide interesting, complementary insights. The time spent *excluding* diagram modifications provides a useful baseline of comparison where the reader is free to consider an effort overhead for model modification that is suitable to her context. The time spent *including* diagram modifications reflect the expected impact of UML on time with our specific modeling tools and given our subjects' training in that particular tool.

Another interesting dependent variable is the functional correctness of the changes, as UML documents may also help achieve better change reliability. Furthermore, because a change can be functionally correct but poorly designed—thus impacting the maintainability of the changed system— we want to assess the design quality of the change design and determine whether UML helps achieve better designs through a better comprehension of the existing system design.

Our experiment therefore has one independent variable (the use of UML documentation) and two treatments (UML, no-UML). It has four dependent variables on which treatments are compared: time to perform the change *excluding* diagram modifications (*T*), time to

perform the change *including* diagram modifications (*T'*), the correctness of the change (*C*), and the quality of the changed design (*Q*). There are also alternative ways to measure the *C* and *Q* variables and this will be further discussed in Sections 3.6 and 3.7.1.

Another important aspect is to decide what should be the baseline against which to compare the use of UML. There are of course an infinite number of possibilities as software development practices are widely varying. But, in our experience, the most common situation can be defined as follows: (1) Source code is the main artifact used to understand a system, (2) Source code is commented to define the meaning of the most complex methods and variables, (3) There exists a high level textual description of the system objectives and functionality. This is therefore what we will use as a basis of comparison in order to determine whether the abstract representations captured by UML helps developers perform their change tasks.

3.2 Experimental Context

Our two experiments took place in significantly different contexts. The Oslo experiment involves 3rd year informatics students which had previously taken two courses with significant UML exposure. The Ottawa experiment involved 4th year Computer/Software engineering students which had previously passed a full term course on UML-based analysis and design and were registered in a second course focusing on UML-based development. A precise comparison of the relevant course credits taken by students in Oslo and Ottawa showed they had comparable programming experience and UML exposure (Section 3.4).

While in Oslo the participants were financially compensated to take part in the experiment, the experiment in Ottawa was part of compulsory course laboratories and the tasks were performed as practical lab exercises. In the latter case we had to ensure that (1) every student would go through the same learning experience, (2) the labs were a valuable practical experience supporting the objectives of the course, and (3) the tasks to perform would be feasible within the scheduled lab hours. These considerations were paramount in selecting the proper course in which to run the experiment and during the

definition of our experiment design and material. Furthermore, they did not know what hypotheses were tested.

3.3 Hypotheses Formulation

The hypotheses for testing the effect of UML documentation on our four dependent variables are given in Table 1: The alternative hypotheses (Ha) state that using UML documents improves three of the dependent variables: less time to complete the tasks when *excluding* diagram modifications, improved functional correctness, and improved design quality. Thus Table 1 defines three of the hypotheses as one-tailed as we expect UML to help people better understand the system design, and hence provide better solutions faster. It is however difficult to have clear expectations regarding the UML effect on time when *including* time spent on diagram modifications (*T'*), as the time overhead to modify the diagrams might be larger than the expected time gains. Thus the hypothesis on time including diagram modifications (*T'* in Table 1) is two-tailed.

Table 1 Tested Hypotheses

Dependent variable	Null hypothesis	Alternative hypothesis
Time excluding diagram modifications	H_0: $T(UML) \geq T(no\text{-}UML)$	H_a: $T(UML) < T(no\text{-}UML)$
Time including diagram modifications	H_0: $T'(UML) = T'(no\text{-}UML)$	H_a: $T'(UML) \neq T'(no\text{-}UML)$
Correctness	H_0: $C(UML) \leq C(no\text{-}UML)$	H_a: $C(UML) > C(no\text{-}UML)$
Quality	H_0: $Q(UML) \leq Q(no\text{-}UML)$	H_a: $Q(UML) > Q(no\text{-}UML)$

When comparing the time spent on tasks across UML and no-UML groups, one should naturally account for the overhead involved in modifying UML diagrams. From this perspective, *T'* is a priori a better measure than *T* when assessing the economic impact of using UML. However, for reasons previously discussed in Section 3.1, it is still relevant to look at *T* as we expect the relative overhead due to model modification to vary a great deal across environments, depending on the specific tools in use and training. Furthermore, as tools are improving their support for model change, we expect this overhead to decrease over time. By reporting results of the impact of UML on *T*, we thus intend to provide a basis for the reader to assess the benefits of using UML based on her own, context-specific estimate regarding the overhead of modifying UML diagrams. In

addition, such results will provide further evidence regarding whether UML, as a minimum requirement, really facilitates code understanding and change.

3.4 Selection of Subjects

Oslo students went through different numbers and types of courses, but all of them had taken at least two OO programming courses and at least two courses where UML was introduced. Carleton students had taken a minimum of two OO programming courses, one course on UML modeling, and were taking a second UML modeling course at the time of the experiment. In both cases, the students were familiar with the tools they had to use (TAU [35] and Visio [21]). Based on this information (summarized in Table 2), we deemed that they had the required training to perform the tasks at a proficient level. In many ways, for the particular tasks involved here, our experience suggests that such students are better trained than most professionals who often have not been formally taught OO design and modeling with the UML, at least not nearly to the same extent. Because students had passed the required courses, we did not perform any selection of subjects in the Ottawa experiment.

Table 2. Summary of competences

	Oslo	Ottawa
OO programming courses (lecture + laboratory)	2 courses: 120 hours	2 courses: 120 hours
Introduction to UML course(s) (lecture + laboratory)	2 software engineering courses introducing UML as one of several topics: Approximately 40 hours of UML-specific training	1 course: 60 hours
Other topics addressed in previous courses	Operating systems, databases, computer architecture, data communication, software engineering	Operating systems, databases, real-time systems, computer architecture

3.5 Experimental Design

3.5.1 Experimental tasks

Both experiments involved the same two systems. The first one is a simple ATM system and the second one is a software system controlling a vending machine serving hot drinks. Both systems were used in previous experiments [3] and were modified for the

experiments we report here. Table 3 provides relevant metrics for the two systems. The systems used were tested before any modifications by devising test suites using the category-partition method testing technique [25]. This test technique was also reused in the Ottawa experiment to assess the functional correctness of the subjects' solutions (Section 3.6) after changes were implemented. The subjects were given a high level textual description of the system objectives and functionality. The source code was also commented to define the meaning of the most complex methods and variables. For all tasks, the subjects were given a precise functional specification illustrated by a test case output that exemplified the details of the requested functionality. We did not provide the subjects with a test harness. They were free to test their changes as they wanted. Note that this was the case of both UML and no-UML groups, so no bias was introduced. The UML documents provided information at a level of details that one would expect at the end of the design phase [11]: a use case diagram, sequence diagrams for each use case, and a class diagram. These correspond to the most commonly used diagrams in practice and we wanted our results to be as realistic as possible. For the same reason, all conditions in sequence diagrams were simply described in English. Example UML diagrams are provided in Appendix D and Appendix E.

Table 3. Details on the two systems (for the first task of each system)

	ATM	Vending Machine
# of classes	15	9
# of operations	33	32
# of attributes	15	7
# of use cases	5	9
# of messages in sequence diagrams	53	74
# LOC	354	297

We defined four tasks in both experiments but they partly differed. The reason is that, after the Oslo experiment, we felt that we needed more difficult tasks to extend the scope of the study. For the Ottawa experiment, one more difficult task was therefore created for each of the ATM and Vending systems. The three most difficult tasks in the Oslo experiment were reused as the easiest tasks in Ottawa. In the remainder of the paper, tasks are referred to according to the task names given in Table 4.

Table 4 Systems' and tasks' Descriptions

System	Tasks	Description	Oslo	Carleton
ATM	Task 1	Print out an account transaction statement	x	x
	Task 2	Transfer money between two accounts	not given	x
Vending	Task 3	Implement a coin return-button	x	not given, already implemented
	Task 4	Make bouillon as a new type of drink	x	not given, already implemented
	Task 5	Check whether all ingredients are available for the selected drink	x	x
	Task 6	Make your own, customized drink based on the available ingredients	"Time sink" task	x

3.5.2 Time Allocation

Oslo students had 8 hours, all in one day, to complete all the four tasks. In addition to the four tasks for which the participants were evaluated, there was a fifth task (Task 6 in Table 4) which was not expected to be completed but was intended to be a "time sink" in which participants could use the remaining time, if any, after the completion of the first four tasks (Tasks 1, 3, 4 and 5 in Table 4). The time sink task was thus included to reduce time ceiling effects: Subjects who work fast may spend more time on the last task than they would otherwise. Similarly, subjects who work slowly may have insufficient time to perform the last task correctly, and may therefore prioritize speed over quality. Consequently, the time-sink task was included as the last change task in this experiment but it was not accounted for in the analysis. Further discussions of the rationale for the time sink task is given in [3].

Carleton students had 5 course laboratories (a week apart) of three hours. The first session was used for exercises and additional training, and each of the four tasks was then performed in subsequent labs. There was no need for a time sink task as students were free to leave the lab once their work was completed.

3.5.3 Other factors to be controlled

As for any software engineering experiment, we expected wide variation in terms of students' ability. In both experiments we used blocking as a means to ensure comparable

skills across student groups using and not using UML, respectively. In Oslo, blocking was based on the number of "passed credits" in computer science specific courses, as this was found to be a good predictor of their performance based on data from a previous experiment with the same systems and tasks [3]. Two blocks were then considered according to whether or not students had passed a minimum of 30 course credits (corresponding roughly to 6 courses): 11 students in the low-credit (below 30 course credits) and high-credit blocks, respectively. The 11 students within each block were then randomly assigned to one of the two treatments. Two students (one from each block, both assigned to the UML treatment), did not show up for the final experiment. Despite the subject loss, the mean number of credits in the resulting groups was still very similar (27.0 for the UML group and 26.5 for the No UML group). In Ottawa, we could use a more direct measure of students' ability by using the grade of their previous OO analysis and design course, which focused on UML modeling. Two blocks were then considered according to whether or not students had obtained a minimum grade of B- in that previous course: 38 students in the low-ability block (grade below B-) and 38 students in the high ability block (grade above or equal to B-).

In Oslo, given the fact that all required technical skills were already acquired in previous courses, students underwent a specific training for the experimental tasks at hand, to get familiar with the experiment support system (Section 3.6), the experimental process, and the development tools. In Ottawa, no experiment support system was used and a very simple tool (Visio), requiring no further training, was used for UML modeling. The first lab was used as a refresher about UML modeling though students all had previously undertaken appropriate courses in OO analysis, design, and programming.

In terms of the systems used, we have indicated that both experiments used the same two systems. In Oslo, all subjects performed all tasks on both systems in the same order, as shown in Table 5. In Ottawa, half the participants started with the Vending Machine whereas the other half worked first on the ATM. The motivation was to make sure that none of the two systems would somehow benefit from learning effects so that they could be distinguished from system effects.

Table 5. Oslo Experiment Design

UML (9)	No UML (11)
Training	Training
ATM Task 1	ATM Task 1
Vending Task 3	Vending Task 3
Vending Task 4	Vending Task 4
Vending Task 5	Vending Task 5
Time Sink (Task 6)	Time Sink (Task 6)

In order to differentiate the treatment (UML) from ordering effects, the Ottawa experiment was designed so that, for each task, groups of (nearly) identical size performed the task with and without UML. In Oslo, this problem was not relevant as half the students were assigned to the no-UML treatment (11 as reported in Table 5) for all tasks and they all performed the same tasks in the same order. The reason why this was possible is that, the students receiving payment for their involvement, there was no ethical necessity to ensure that all of them would go through the same learning experience. In a course lab context, this was of course an absolute requirement.

The exchange of information among students was prevented, both during or in-between labs. In Oslo, students were paid and were aware this was an experiment for which they were supposed to work individually. They were also monitored by several researchers during the experiment. In Ottawa, students were graded based on the resulting quality of their tasks and were told to work individually. Their work was carefully monitored during all labs and, since they were not aware beforehand of the precise plans for all five labs, they had no reason to suspect they would work on identical tasks. Furthermore, the material necessary for performing the tasks was not available between labs.

The experimental design for the Ottawa experiment is summarized in Table 6, where each group involves (approximately) the same number of participants (e.g., 22 in group 1) and is randomly assigned students according to the blocking procedure described above. Table 6 shows what task was performed, on which system, and in which order for each of the four groups in the Ottawa experiment. We can clearly see that the number of people performing tasks on the ATM first is equal to the number of people working first

on the Vending Machine. Similarly, the number of participants working first with UML and without UML is approximately the same. Hence the effects of individual capabilities, system differences, and the order in which UML is used are counterbalanced.

The groups are not exactly of the same size due to subject loss (i.e., people missing labs, being sick, etc.) after subjects were assigned to the groups. However, a one-way analysis of variance clearly shows that the grade means are not significantly different across groups.

Table 6 Ottawa Experiment Design

	Group 1 (22)	Group 2 (17)	Group 3 (21)	Group 4 (18)
Lab 1	ATM Task 1	ATM + UML Task 1	Vending Task 5	Vending + UML Task 5
Lab 2	ATM Task 2	ATM + UML Task 2	Vending Task 6	Vending+ UML Task 6
Lab 3	Vending + UML Task 5	Vending Task 5	ATM + UML Task 1	ATM Task 1
Lab 4	Vending+ UML Task 6	Vending Task 6	ATM + UML Task 2	ATM Task 2

3.6 Instrumentation and Measurement

In Oslo, the data collection and the logistics of the experiment were supported through a web-based experiment support system (SESE) [4]: systems and task descriptions were distributed, time was measured, task solutions and a comprehensive set of qualitative data pertaining to each task were collected through SESE. In Ottawa, since that system could not be used in ordinary lab settings, students had to download documents from a web site and were asked to send their solutions by email as soon as they were completed. In addition, after each task, students were asked to fill a survey questionnaire to collect data about their perceived difficulty of the task, their experience and familiarity with the tools, and whether they thought UML was useful (Appendix B).

In Oslo, test cases were devised to perform basic functionality testing of the changes, i.e., test the main scenario of the changed function. All task solutions were also inspected manually to further assess the “degree” of correctness. On the basis of the functional

testing and the manual inspection, the solution to each change was graded on a 6-point scale to indicate the amount of work required to fix any deviations from the prescribed functionality, as shown in Table 7. However, based on the feedback we received from the rater using this scale, doubts were raised about the reliability of scores below 5. The top end of the scale (5 and 6) is more reliable (less subjective) because it is mainly based on the results of the functionality testing: solutions with no or only cosmetic differences from the expected output were considered correct. As a result, our analysis relied on a binary classification of correctness where only solutions with scores 5 or 6 were considered correct as cosmetic variations in the expected output were not considered to be relevant.

Table 7 Coding scheme of the correctness measure.

Correctness	Interpretation
6	Correct solution, passed the test case
5	Cosmetic differences in the output. Trivial to fix
4	Small logical errors that are estimated to be very simple to fix
3	Some errors that are estimated to take some time to fix
2	Incomplete solution that are estimated to take a long time to fix
1	Very incomplete solution or no solution delivered

In addition, through SESE, the students answered a questionnaire after each task, and had to provide feedback every 15 minutes on what they were doing, thus providing continuous qualitative insight during the experiment. At the end of the experiment, semi-structured interviews of the participants took place and were analyzed using the QSR N6 qualitative analysis tool [31], as described in Section 3.7.2.

In Ottawa, because we were dealing with a larger number of subjects than in the Oslo experiment, we had to automate the testing procedure as much as possible. For that purpose we specified a precise functional test plan for each change, based on a black-box test technique called Category-partition [25]. Only failed test cases were then inspected manually in order to determine whether the failure was due to minor cosmetic differences in the output or a functionally incorrect change. An additional motivation of this test suite-based strategy was to have a finer granularity and more objective correctness measurement than the binary correctness evaluation used in the Oslo experiment. This

was especially important as the emphasis of the Ottawa experiment was more on quantitative analysis.

We also wanted to assess the design quality of the proposed solution, independently of the functional correctness of the change. For each change, we performed a precise analysis of all solutions that could be considered proper based on standard design strategies for class responsibility assignment [11][1]. We then counted the number of operations, attributes that should be added, modified, or deleted based on each identified solution. The design quality of a task solution was then assessed on that basis by counting the number of elements that were correctly and erroneously added, changed, and deleted[2]. For the sake of simplification, we combined additions, deletions and changes into a single count thus yielding two counts *Q* and *Q'* for correct and erroneous changes, respectively. If we take Task 6 as an example, it involved the addition of two methods, plus the change of one method and one constructor. For the sake of the example, we label them *m1*, *m2*, *m3*, and *c*, respectively. A solution that would correctly add *m1* and change *m3*, but would forget *m2* and the change in *c*, would result into two correctly added/changed class elements (Q = 2) out of a maximum of four. Furthermore, if we assume that the evaluated solution manages to implement the functionality through a substandard design modifying one method different from *c* and adding one method different from *m2* (e.g., in a different class), then this would result into two erroneously added/changed class elements (Q' = 2). It is even conceivable that solutions modify, add, or delete the right elements but alter other elements as well. In that case, we would obtain a perfect *Q* score but a suboptimal *Q'* score (Q' > 0). Both *Q* and *Q'* counts are deemed relevant as they offer complementary measurements of design quality, which cannot meaningfully be combined into one measure. *Q* quantifies the extent to which a design solution complies with expected changes in the design and *Q'* quantifies the extent to which unnecessary, suboptimal changes are performed.

[1] In practice, there are usually a few such solutions. In our case, there are 2 of them for task 6 and only one for the other tasks.

[2] Note that in general when applying such a strategy, one has to identify the correct solution that is closest to each evaluated subject solution. However, to obtain comparable measurements, differences in model elements involved have to be accounted for. This was not an issue in our case since there was only one proper solution for all tasks except Task 6. Moreover, the two proper solutions for Task 6 involve exactly the same numbers of classes, operations, and attributes (details are provided in Appendix A).

3.7 Analysis procedure

3.7.1 Quantitative Analysis

Recall that the experiment has one independent variable (the use of UML documentation) and two treatments (UML, no-UML). The Oslo experiment has three dependent variables: the time to perform the change task excluding (*T*) and including (*T'*) diagram modifications, and a binary correctness score (*C*). The Ottawa experiment considered four dependent variables: *T* and *T'* again, a ratio scale functional correctness measure counting the number of passed test cases (denoted *C'* to differentiate it from *C*) and design quality measured in two distinct and complementary ways: The number of *correctly* changed elements (*Q*), and the number of *incorrectly* changed elements (*Q'*). Analyzing both *Q* and *Q'* will provide complementary insights in terms of the degrees to which required changes and unnecessary changes were performed. The level of significance for the hypotheses tests are set to $\alpha = 0.05$. The reader should however keep in mind that we perform multiple tests and, in order to allow for a stricter and more conservative interpretation of the results, we will provide p-values.

In the Oslo experiment, only univariate analyses of the dependent variables are performed to test the hypotheses, both for each task individually and across all tasks. For the time dependent variables (*T* and *T'*), two-sample *t*-tests are performed [12]. Additionally, to reduce potential threats to validity resulting from violations of the *t*-test assumptions, non-parametric Wilcoxon rank sum tests are also performed [12]. However, overall the two alternative tests yielded very consistent results and thus only the *t*-test results are reported unless inconsistencies are present. A likelihood ratio Chi-Square test [12] is used to test the difference in *proportion of subjects* with correct solutions for each individual task. Furthermore, a one-sided, two-sample *t*-test is performed to test the difference in correctness between UML and no-UML subjects *across* all four tasks, by first calculating each subject's percentage of correct solutions based on the binary correctness score for each task (C).

In the Ottawa experiment, both univariate and multivariate analyses were performed. The time dependent variables (*T* and *T'*), correctness (*C'*) and design quality (*Q* and *Q'*) are

analyzed for each individual task by comparing the results of the two groups of subjects that received UML for a given task with the two groups of subjects that did not receive UML for that same task. As in the Oslo experiment, both two sample *t*-tests and the non-parametric equivalent Wilcoxon rank sum tests are used to perform the univariate tests of the hypotheses.

The Ottawa experiment involved more subjects and a more complex design than had the Oslo experiment, hence both enabling and motivating the use of more sophisticated statistical analyses that not only account for UML effects, but also ordering and subject ability effects at the same time. Thus we performed a 3-way analysis of variance (ANOVA) [12] to test the simultaneous effect of *UML*, *Task Order* (to assess cross-over effects), *Block* (to assess ability effects) and the interactions between these factors. The reason why this 3-way ANOVA is better than, say, a simple two-sample *t*-test on the dependent variable is two-fold. First, this allows us to investigate interaction effects between *Block* (ability) and the use of UML. Perhaps UML has a different effect on subjects with different ability. Second, by including *Block* and *Order* (and possible interactions) in the model, the variation they explain in the data is removed and the analysis of the effect of UML becomes as a result statistically more powerful, e.g., we are more likely to see any significant effect if there is one. Based on the ANOVA results, we also calculate the least-square means [13] for the UML factor, to visualize the corresponding effect size for each dependent variable. Least squares means (LSM) are predicted values from the ANOVA model across the levels of a categorical independent variable where the other model factors are controlled by being set to *neutral* values (based on predicted values for all levels). The LSM plots show both the means for each level and 95% confidence intervals.

Note that the 3-way ANOVAs are performed for each individual task. Another possibility would be to perform one overall, *4-way* ANOVA for all subjects and tasks, where *Task* thus would be a forth factor in the model. However, given that there are four observations per subject (one per task), the most critical underlying assumption of ANOVA – independence between observations – would be violated in this case since within-subject observations are likely to be correlated [22]. A proper 4-way analysis of the factors

would thus necessitate the use of statistical techniques that account for this correlation, such as Generalized Equating Equations (GEE) [22]. However, based on the preliminary, descriptive statistics it became evident that there are strong interaction effects between the factors Task and UML. Consequently, the use of GEE or similar methods was not necessary given our data; the presence of strong interaction effects justify that we nevertheless should study the effects of UML for each individual task, in which case the aforementioned 3-way ANOVA is appropriate. Note that the ANOVA procedure is based on a number of assumptions in addition to the one of independent observations. Each combination of factor levels (e.g., UML and Block) has to be assigned subjects in a random fashion and should also have an equal (or similar) number of observations (subjects). The dependent variable should be, for each level combination, normally distributed and the variance should be equal. Obviously, the last two assumptions are usually not met in practice but ANOVA is known to be very robust to departures from these assumptions [15], especially when the sample sizes are equal or similar for each level combination.

3.7.2 Qualitative Analysis

In the Oslo experiment, qualitative data were collected from three sources: Change task questionnaire comments, think-aloud comments, and interviews. 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. 76 out of a total of 120 questionnaire comments were actually filled out and the information stored in a database. Additionally, the SESE tool polled the subjects for feedback during the experiment (details on the feedback collection method are provided in [17]). 225 such comments were collected and stored in a database during the experiment.

Within one week of the main phase of the experiment, semi-structured interviews were conducted with 19 of the 20 subjects (one subject did not show up for the interview). An interview guide with relatively open questions was prepared. The interviews were recorded on tape. Each interview lasted about 12 to 35 minutes, depending on the group

to which the subjects were assigned 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 group used the diagrams in order to understand the program and several questions were related to this. Consequently, the interview guide was extended mid-way in the interview schedule, when it became evident that many subjects did not use the documentation in the expected way. The new interview guide (Appendix C) made sure that we would not forget to ask questions with regards to why they did *not* use the UML diagrams the way we had anticipated. To increase the accuracy and comprehensiveness of the analysis, the recorded interviews were carefully transcribed.

The qualitative data coming from three sources (task questionnaires, feedback data and transcribed interviews) were analyzed using QSR N6 [31], a tool for qualitative data analysis. Using the tool, the texts were examined, sorted and categorized into different concepts and a tree structure with concepts and sub-concepts was built on the basis of the data. The analysis attempted to assess how UML was used and the subjects' perceptions of the costs and benefits of using it. It also analyzed how the subjects were working and what types of problems they were experiencing on the different tasks, to better understand how the access to UML documentation made a difference. To avoid introducing a bias in the interpretation of the qualitative data, the initial qualitative analysis was performed by one of the researchers who at the time had no knowledge of the quantitative results. After the quantitative data was analyzed, a second round of qualitative analysis was performed in an attempt to better explain the quantitative results, as further explained in Section 4.4.

The Ottawa experiment did not involve any sophisticated qualitative analysis like in Oslo. However, as discussed in Section 3.6, participants filled out questionnaires after each lab regarding the task and, when relevant, the use of UML. Standard techniques of phrasing subjective questions and designing survey questionnaires were followed [24], so as to avoid bias and to increase reliability of responses. A one-way analysis of variance was used to analyze those data

4 EXPERIMENTAL RESULTS

Recall that we will test four hypotheses with regards to the effect of UML on (1) the time to perform the change task excluding diagram modification (using the variable *T*), (2) the time to perform the change task including diagram modification (using the variable *T'*), (3) functional correctness (using the variable *C* in the Oslo experiment and *C'* in the Ottawa experiment), and (4) design quality (using the two complementary quality measures *Q* and *Q'*).

4.1 Descriptive Statistics

The descriptive statistics of the Oslo experiment are given in Table 8. The results indicate that the subjects receiving UML documentation spent on average 25 percent less time (*T*) to solve the tasks than did the subjects without UML documentation (e.g., a median of 15 compared to a median of 20 for Task 3). One exception is Task 5, for which subjects without UML perform the task slightly faster than subjects who received UML documentation. Furthermore, the variance is much lower for the subjects who received UML documentation, with much lower maximum values for all tasks (e.g., 95 instead of 150 for Task 5). Overall, a larger portion of subjects produced correct solutions in the UML group. This is particularly true for Task 5: 46% and 89% of the answers were correct without UML and with UML, respectively. Overall, when accounting for model modification (*T'*), subjects working without UML models spend less time on the tasks. Again, this is especially true for the last, most complicated one (Task 5).

Table 8 Oslo: Descriptive statistics per task

		Tasks											
		Task 1			Task 3			Task 4			Task 5		
Dep. Var.	Group	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
T (excl. model modification)	No UML	20	75	240	5	20	60	10	22	55	16	54	150
	UML	31	53	95	8	15	23	12	19	35	45	65	95
T' (incl. model modification)	UML	43	70	105	15	27	41	24	36	85	62	101	132
C (binary)	No UML	46%			91%			91%			46%		
	UML	56%			89%			100%			89%		

The descriptive statistics of the Ottawa experiment are given in Table 9. The results indicate that, for the common tasks (1&5), results are consistent with those obtained in Oslo with respect to time: UML leads to increased time when accounting for the modification of models (*T'*). When considering only code modification time (*T*), the UML subjects seem to take more time for all tasks but Task 6. However, functional correctness (*C'*) does not seem to improve when using UML as opposed to what was observed in Oslo. Task 2 also shows a significant increase in time for UML subjects, especially when accounting for model modifications. Correctness does not seem to increase though. For Task 6, we obtain a very different picture as time does not significantly increase—even when accounting for model modifications—whereas functional correctness (*C'*) clearly increases (the median increases from 0/12 to 5/12). With respect to design quality (correct changes), UML seems to have a positive impact in Task 1 (max), Task 2 (min), and Task 6 (min and med), but only a thorough statistical analysis will tell with certainty. Similarly, for incorrect changes, the statistics of tasks 1 and 6 suggest a difference between UML and no-UML groups. Note that our discussion at this point is informal and that the effect size of UML will be more precisely discussed when presenting multivariate analysis results.

Table 9 Ottawa: Descriptive statistics per task

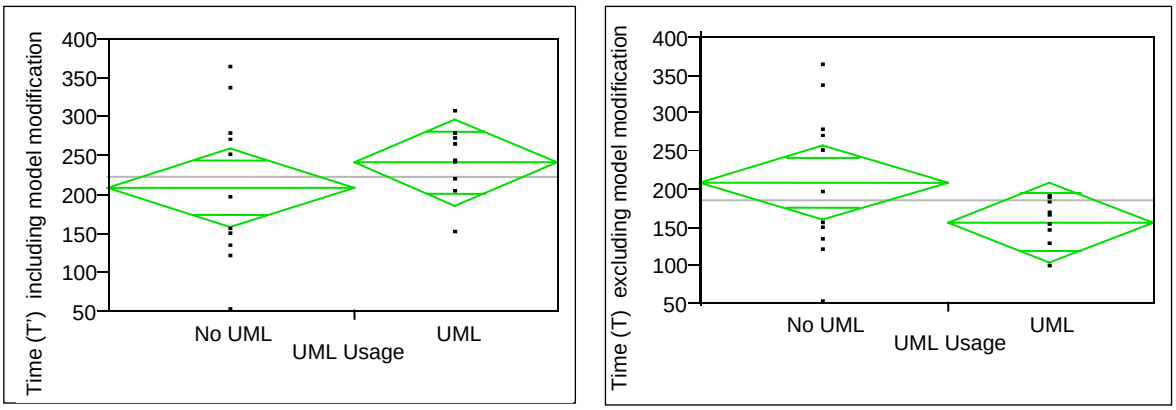
		Tasks											
		Task 1			Task 2			Task 5			Task 6		
Dep. Var.	Group	Min	Med	Max	Min	Med	Max	Min	Med	Max	Min	Med	Max
T (excl. model modification)	No UML	22	66.5	159	20	75	154	32	89.5	180	74	166.5	194
	UML	24	82	240	38	87	162	35	99	196	51	141	184
T' (incl. model modification)	UML	40	128	261	74	149.5	180	58	141	223	75	168	184
C' (ratio)	No UML	2/8	8/8	8/8	0/8	5/8	8/8	0/8	5/5	5/5	0/12	0/12	12/12
	UML	2/8	8/8	8/8	2/8	5/8	7/8	1/8	5/5	5/5	0/12	5/12	12/12
Q (Correct changes)	No UML	1/9	4/9	4/9	0/8	8/8	8/8	1/4	4/4	4/4	0/4	1/4	4/4
	UML	1/9	4/9	9/9	4/8	8/8	8/8	1/4	4/4	4/4	1/4	3/4	4/4
Q' (Incorrect changes)	No UML	0	0	7	0	0	1	0	0	1	0	0	10
	UML	0	0	3	0	0	0	0	0	2	0	0	1

4.2 Univariate Analysis

4.2.1 Oslo Experiment

For the Oslo experiment, univariate results for time are illustrated in Figure 1(a) and 1(b). Figure 1(a) shows means diamonds[3] of the total time spent on all four tasks when including diagram modification (*T'*) and we can see that UML subjects spent, on average, more time than their no-UML counterpart but the difference is not statistically significant. If we do not account for the time spent modifying the diagram (Figure 1(b)), then the difference is larger, and in the opposite direction, but still not significant. The lack of significance may be partly due to the relatively small number of participants. In any case, it suggests that when including the modification of models (diagrams) UML does not seem to have provided an advantage in Oslo. A more detailed analysis for each task (which is not reported here) yields similar results: no significant difference. In terms of correctness, Figure 2 shows that no practically significant difference is visible for the first three tasks. For Task 5, however, the percentage of no-UML subjects correctly implementing the task is nearly half that of UML subjects. A Likelihood Ratio Chi-Square test confirms that the difference in proportion is significant at $\alpha = 0.05$ (p-value = 0.034). This result is likely due to the fact that Task 5 is much more difficult than the other three ones. Furthermore, Figure 3 shows the percentage of correct solutions across all 4 tasks and we can clearly see UML subjects perform better. However, because of the small sample at hand, a two-sample t-test yields a p-value slightly above 0.05.

3 This was explained in Section 3.7



(a) Including diagram modification (b) Excluding diagram modification

Figure 1 Oslo: Time to complete all four tasks (including or excluding diagram modification)

Figure 2 Oslo: Percent of subjects with correct solutions

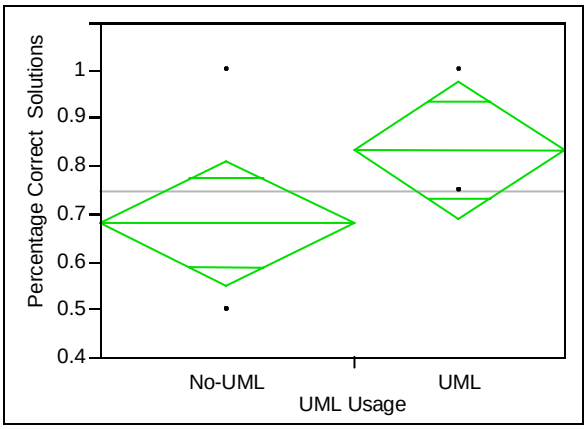


Figure 3 Oslo: Percentage of correct solutions

4.2.2 Ottawa Experiment

In the Ottawa experiment, in order to make the time comparisons more meaningful, we decided not to account for solutions that did not compile. We thought this was a minimal quality requirement for the time data to be comparable. Figure 4 shows the time (*T*) distributions for each of the four tasks, clearly suggesting that tasks are increasingly more difficult, especially Task 6[4]. In this experiment we look at each of the tasks individually since, as we will see, different results will be observed due in part to the variation in complexity. When not accounting for model modification time (*T*), though UML subjects take on average more time for the first three tasks and less time for the fourth task than their no-UML counterpart, none of the differences are significant at a $\alpha = 0.05$. The results, as suggested in the previous section, are very different when accounting for model modification (*T'*): for the first three tasks the UML subjects show significantly higher time values with p-values equal or below to 0.003 (two-tailed t-test and non-parametric Wilcoxon (rank sums) test[5]).

During our monitoring of the labs, we noticed that some people were not using the UML models though they were provided to them. Furthermore, some people never returned the

[4] Recall that, due to our design, this is not an ordering effect.

[5] From now on, when mentioning a t-test result, the reader will assume that the equivalent, non-parametric Wilcoxon (rank sums) test yields an equivalent result unless otherwise mentioned.

modified UML models. We considered these cases suspicious as once you have used the model to determine the changes to be performed, it is then easy to modify the diagrams. We therefore thought that not returning the modified UML diagrams was an indication that these diagrams were probably not used to a great extent or not at all. Further interviews with the concerned students confirmed that they mostly reverted to their old reflexes of relying on code.

Because there was no practically significant difference between UML and no-UML groups for the first three tasks, we focus here on the results of Task 6 after removing the 11 students who did not return their modified UML diagrams. In order to check whether we had introduced a new threat to internal validity (removing subjects may not be an entirely random process), we checked whether after removing these 11 students, the grade distributions were still comparable across UML and no-UML subjects. Figure 5(b) confirms this is the case.

We can see from Figure 5(a), that the average time spent by UML subjects on Task 6 is significantly lower than non-UML subjects: an average difference of over 20 minutes (on an average time of 145 minutes). A t-test clearly shows the difference is statistically significant (p-value = 0.0018). In other words, when participants actually used the UML models for change impact analysis, this resulted in saving time when changing the code.

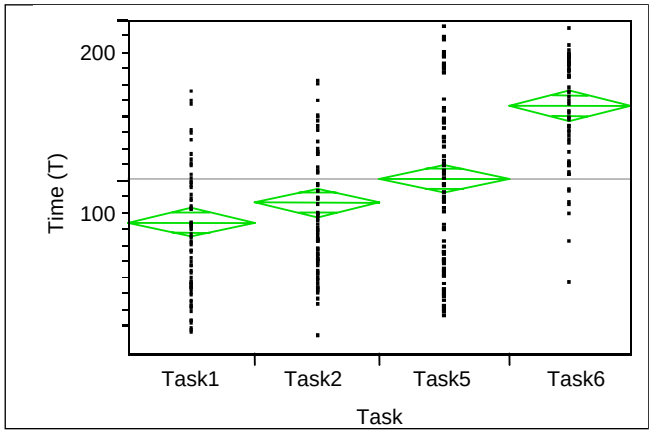
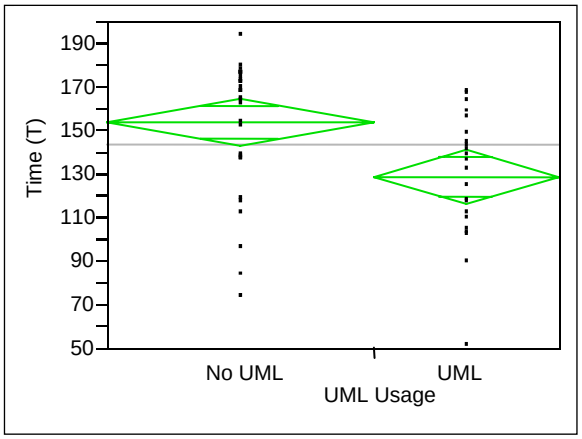
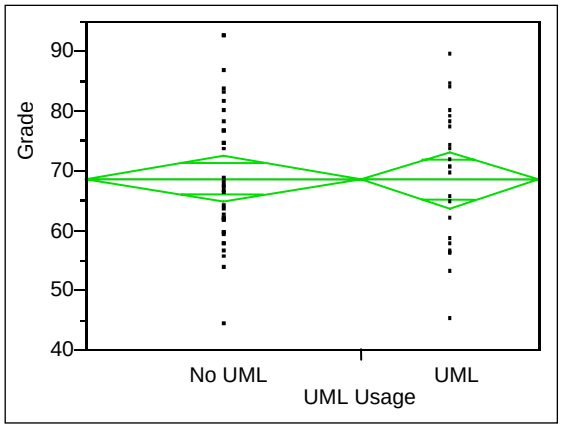


Figure 4 Ottawa: Time (*T*) to complete each of the four tasks



(a) Time (*T*) to complete the task (excluding time to modify UML diagrams)



(b) Grade distribution

Figure 5 Task 6 – excluding cases where no modified diagrams were returned

But if we add the time required to change the UML model to the time required to perform the code change, then, even for Task 6 (Vending machine), there is no time difference between the UML and no-UML subjects (Figure 6).

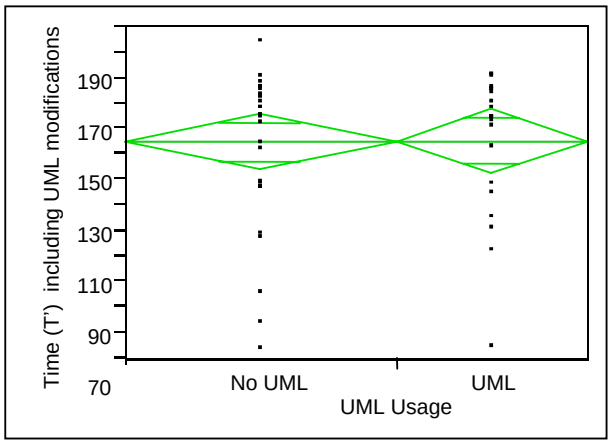


Figure 6 Total time to complete Task 6 – including time to modify the UML diagrams

Let us now turn our attention to correctness. We first measured correctness as the number of successful test cases for each change. These test cases, as mentioned in Section 3.6, are functional test cases derived in a systematic way using the Category-Partition strategy [25], one of the most well-known black-box test techniques. For each change, we tried to devise the best test suites possible. But since the changes are not that extensive, those test

suites remain rather small (ranging from 5 to 12 test cases). One issue though was to decide how to handle solutions that were not compiling and therefore not testable. We decided that such solutions should be assigned the lowest score, that is, zero, since no test case could be run successfully. As for time analysis, and for the same reasons, we decided to leave out the cases where no modified model was provided. Results showed that there was no significant difference between UML and no-UML subjects for all tasks but Task 6. Figure 7 clearly shows a practically significant difference in the correctness distributions: an average difference of 2.25 between the two treatments. A one-tailed t-test shows this difference is significant at $\alpha = 0.05$ (p-value = 0.041).

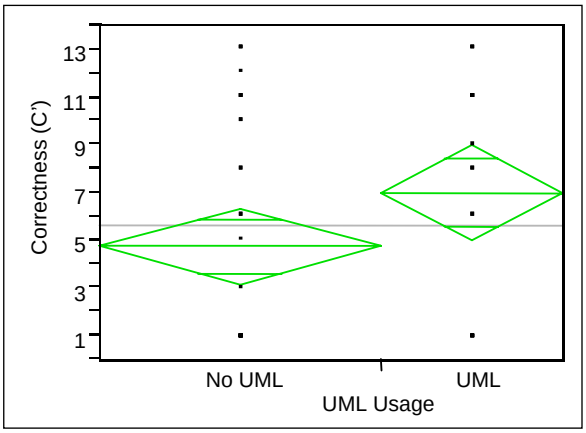


Figure 7. Correctness for Task 6

Another dependent variable we considered in the Ottawa experiment was the design quality of solutions. As described in Section 3.6, we identified, for each change, alternative proper design solutions according to standard ways to distribute responsibility across classes [11]. For each solution, recall we identified the class features that needed to be changed, deleted, and added. Design quality was then measured as a percentage of correctly changed, deleted, and added features. When comparing design quality of UML and no-UML solutions, once again, a significant difference was observed only for Task 6. All the features concerned by the change for Task 6 can be found in Appendix A, where two alternative acceptable solutions are considered. For each provided solution, the authors went through the code and compared it with either of the two alternative solutions, whichever was closest. Figure 8 (a) shows clearly that UML solutions have a

better design quality in terms of correctly changed elements (the average difference is 0.50 on a 5 point scale). This difference is statistically significant (p-value = 0.032) when running a one-tailed t-test. When looking this time at the number of incorrectly changed elements, which is the other relevant design quality measure, we also see (Figure 8 (b)) a practically (average difference of 1.34) and statistically significance difference (p-value = 0.0043). Even if we look at it in a binary way by analyzing proportions, we see that when not using UML, roughly 41% of the solutions contain at least one incorrect change whereas none of the UML solutions do. A likelihood ratio chi-square test for proportions shows this is statistically significant (p-value < 0.001).

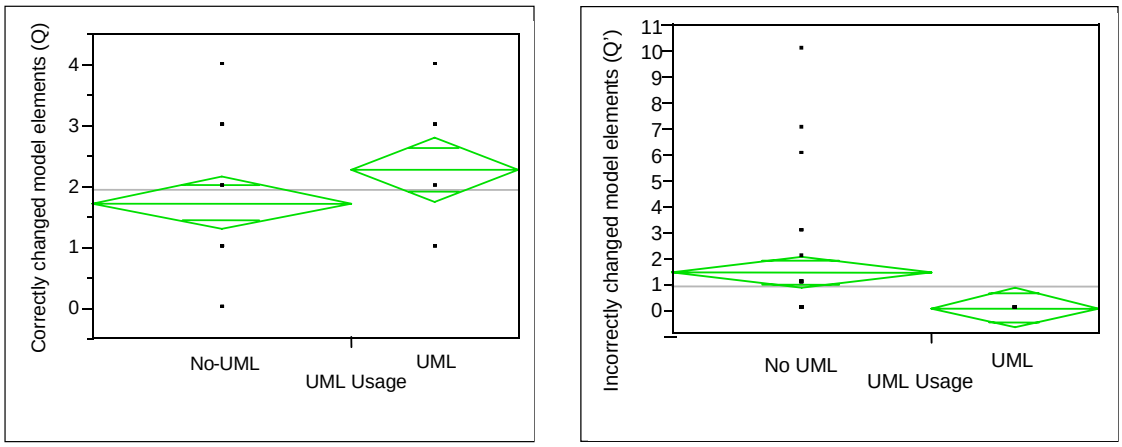


Figure 8 Design quality for Task 6

4.2.3 Summary

As far as time is concerned, UML does not seem to provide an advantage when accounting for the additional time needed to modify models. Even when we do not account for the modification of models, UML subjects do not appear to be faster for the first three tasks of the Ottawa experiment. Tool support for model-code consistency could, however, be improved by providing functionalities to keep models and code in sync. Certain tools, such as Together from Borland [8], have already started to do so and we expect this trend to continue in the future.

In terms of correctness—though the two experiments use different correctness measurement (section 3.6)—both experiments show that, for the most complex task,

UML subjects perform significantly better than no-UML subjects. There are two main plausible ways to interpret these results. Because of learning effects, subjects really benefit from using UML after they performed the first three tasks (this is only true for half the subjects in the Ottawa experiment). Another possibility is that they only benefit for the most complex task, i.e., Task 5 and Task 6 for the Oslo and Ottawa experiments, respectively. However, recall that the most complex Oslo task is the second most complex Ottawa one. If our latter hypothesis is true, why do we not obtain consistent results in the Ottawa experiment for Task 5? It is therefore plausible that what we observe is the compounded result of high task complexity and learning effects.

A similar comment can be made for our measures of the design quality of subjects' solutions. UML subjects provide significantly better quality designs for the most complex task (Task 6) of the Ottawa experiment and this is likely to be due to the fact that UML models help better understand the existing design and therefore help the subjects making appropriate design decisions for the most complex task. For the simpler task, UML is probably not necessary.

4.3 Multivariate Analysis

In Oslo, because of the limited size of the data set, only a univariate analysis was possible. The advantage of multivariate analysis is that we can account for not only UML effects, but also ordering and subject ability effects at the same time. This may lead to more significant results and help assess how strong the UML effects are relative to the other factors.

The ANOVA results for code change time (*T*) for Task 6 are presented in Table 10. Recall, for reasons explained in Section 4.2, that we have removed non-compiling solutions as well as cases where no modified UML model was provided. We can see that both *UML* (consistent with univariate analysis) and the order in which the tasks were performed (*Order*) are significant but that people's ability as measured by the block (*Block*) they were assigned to (based on the previous modeling course grades) is not statistically significant. Both *Order* and *UML* show a similar level of significance (p-value) and, based on their estimated parameter, the effect of *UML* is equivalent to that of

Order, which captures learning effects. Based on the Least-squares means (see Section 3.7) of the ANOVA model (Figure 9), UML helps save about 25 minutes on average.

Note that if we include model modification time (*T'*), then *UML* has no significant effect again. This confirms that without better tool support to change models, UML does not yield time savings.

Table 10 ANOVA Results for Time (T), Task 6

Source	DF	Parameter estimate	Sum of Squares	F Ratio	Prob > F
UML	1	13.96	9587.85	13.14	**0.0007**
Order	1	12.56	7891.12	10.81	**0.0019**
Block	1	-5.23	1342.21	1.84	0.1814
Residual	47		34284.95		
Total	50		52076.15		

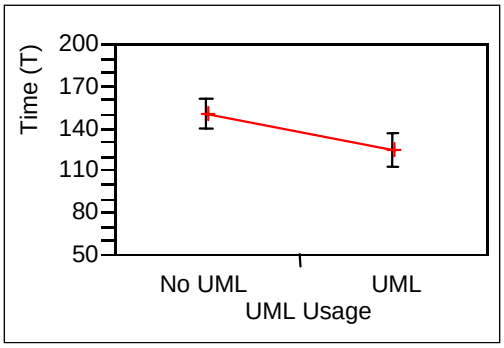


Figure 9 Least-square means for ANOVA time analysis, UML usage

In terms of correctness (*C'*), ANOVA results (Table 11) show that both *UML* and *Block* are statistically significant. There seems to be no ordering effects though for correctness. Recall that we left out solutions for which no modified UML model was provided. Based on parameter estimates we see that *UML* effects are similar to *Block* effects, that is, the impact of student ability on correctness. The Least-squares means plot (Figure 10) show that, on average, UML helps increase correctness by nearly 3 test cases (2.7), which represent close to 25% of the total number of test cases.

Table 11 ANOVA Results for Correctness (C'), Task 6

Source	DF	Parameter estimate	Sum of Squares	F Ratio	Prob > F
UML	1	-1.33	94.89	4.49	**0.0388**
Order	1	-0.26	3.90	0.18	0.6691
Block	1	1.31	93.91	4.44	**0.0398**
Residual	48		989.03		
Total	52		1098.40		

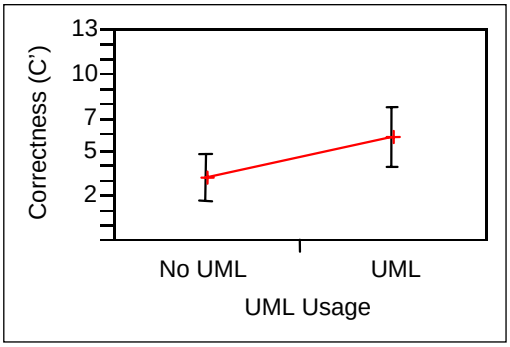


Figure 10 Least-square means for ANOVA correctness analysis, UML usage

We now turn our attention to design quality. The first analysis performs a three-way ANOVA using *UML*, *Block*, and *Order* as factors and the number of incorrectly changed model elements (*Q'*) as dependent variable. Results in Table 12 show that both *UML* and *Order* are significant at $\alpha = 0.05$, as for the time (*T*) dependent variable in Table 10. However, note that *Order* has a negative effect: participants introduced more incorrect changes when Task 6 was performed in lab 4. This might be explained by a loss of motivation as people perform their fourth lab and reach the end of the academic term. The Least-squares means plot (Figure 11) shows that, on average, UML helps reduce incorrect changes by 1.

Table 12 ANOVA Results for Incorrectly changed model elements (Q'), Task 6

Source	DF	Parameter estimate	Sum of Squares	F Ratio	Prob > F
UML	1	0.67	23.57	2.60	**0.01**
Order	1	-0.51	14.89	-2.07	**0.04**
Block	1	-0.04	0.10	0.17	0.86
Residual	48		169.55		
Total	52		180.86		

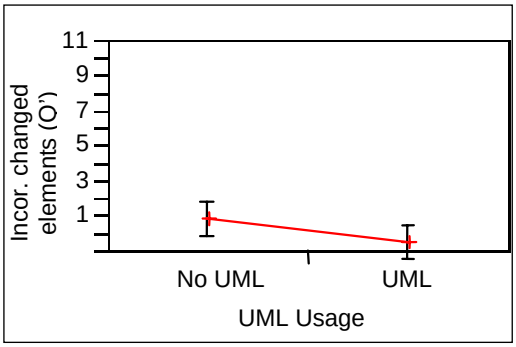


Figure 11 Least-square means for ANOVA design quality analysis (Incorrectly changed elements), UML usage

When using the number of correctly changed element (Q) as a design quality measure, none of the three factors are significant at $\alpha = 0.05$. Since univariate analysis also showed a borderline level of significance, this result is no fully surprising.

The analysis above only looks at the main effects of independent variables on four dependent variables. Though we have also analyzed possible interaction effects between UML and the two other variables, no such effect appeared to be significant.

In terms of the usual ANOVA assumptions:

- Usual statistical tests for unequal variance [15] have shown, for all three factors and all dependent variables, that the variance is not significantly difference across levels.

- The distributions of time, correctness, and quality are not normal. However, the dependent variable distributions do not show extreme outliers or multi-modal

distributions and as discussed in Section 3.7, ANOVA is known to be very robust to departures from the normality assumption [15] when the samples sizes are equal or similar for each level combination, which is the case in our experimental design.

4.4 Qualitative Analysis

In Oslo, based on interviews, the questionnaire comments, and the "think aloud" feedback comments (Section 3.7.2), the qualitative analysis yielded the results presented in Table 13 (one subject in the UML group did not participate in the interview). The results suggest that the extent of use of the UML documentation and its impact varied among the UML subjects. The experiment required that all subjects update the diagrams before they moved on to the next task. However, the extent to which they used the UML models to identify change locations prior to performing code modifications greatly varied among subjects. Seven of the nine subjects assigned to the UML group updated the UML documentation after they completed the coding. Only two subjects used the UML documents actively during the development, e.g., to identify change locations. Some complained about the updating of the diagrams because the diagrams were very large and difficult to update, in particular the sequence diagrams. Several expressed dissatisfaction with TAU, despite having extensive training in that tool. This led us in the Ottawa experiment to use a simpler tool (Visio). A general consensus among the subjects was that the diagrams were more useful on more complex tasks and that they often were superfluous on easier tasks. This observation in the Oslo experiment was what made us be suspicious of solutions without modified diagrams in the Ottawa experiment and led to our decision above to leave such observations out of the analysis. Since the education system in both locations is rather similar in terms of courses and the order according to which they are delivered (i.e., emphasis on coding, modeling mostly coming in later years), we can therefore expect the issues regarding the usage of UML diagrams in Oslo to be also relevant for the Ottawa experiment.

The subjects who claimed not to have used UML models when they could have (but instead just updated them after coding) provided the following justifications:

- They were more used to look at the code than diagrams. This is not surprising given typical computer science curricula where people start learning how to code far before they see a first model.

- They felt that analyzing the documentation required too much time. This is a typical problem in many development organizations where developers and managers perceive (consciously or not) any non-coding task as wasteful.

- The tasks were relatively small; therefore it was sufficient to rely on the code.

Table 13 Summary of qualitative results (the numbers in parentheses indicate the proportion of subjects for which the statement applies).

Topics	UML	No UML
The use of UML documentation	• The UML documentation was used to locate code changes (2/9) • The documentation was used a little prior to code changes (3/9) • The documentation was ignored prior to code changes (4/9) • The documentation was updated after the code changes (7/9)	• UML diagrams were drawn by hand to help solving the most difficult tasks (4/10)
The benefit of UML documentation	• UML useful for solving the tasks by providing overview and structure (2/9) • UML useful for control of code changes and overview to some degree (3/9) • UML not useful for the task solving (4/9)	• UML would have been useful to locate the code changes (8/10) • UML would have been helpful with how to do the code changes (4/10) • UML would not have been useful on the easiest tasks (15/20*) • UML documentation would have improved the solution (3/10)
Time	• Time pressure at the end of the experiment (8/9)	• Time pressure at the end of the experiment (4/10) • There was sufficient time for task solving (6/10)
Problems	• It was bothersome and difficult to update the UML diagrams (3/9) • Dissatisfaction with Tau_UML (6/9) • Java-related problems in the beginning of the experiment (2/4***) • Problems with getting an overview of the last task (4/9) • Out of practice with Java programming (2/9)	• Java-related problems in the beginning of the experiment (4/6***) • Out of practice with Java programming (7/10) • Problems with getting an overview of the program (10/10**)

(* = the statement is a combination of two questions
** = the statement is based on questionnaire comments
*** = the statement is based on think aloud comments)

Nevertheless, half of the UML subjects considered UML useful for obtaining an overview of the code, although there was a wide variation in terms of perceived benefits. Four of the UML subjects still thought it was difficult to get a sufficient overview of the program to perform the last, most difficult task. On the other hand, a common statement by the no-UML subjects was that it was difficult to get an overview of the programs, not only for the most difficult task, but in general. This was especially evident in the questionnaire comments, but also to a certain extent in the interviews and the think aloud comments. Some no-UML subjects even drew their own diagrams when they solved the more difficult tasks, in particular Vending Task 5. Several thought that UML diagrams would have helped them to locate 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 complex tasks.

Another striking result that we believe is related to the above is that seven out of ten no-UML subjects said that they were out of practice with programming/Java, whereas only two out of nine UML subjects did. Counter to this, due to the randomized blocking scheme, the subjects in the two groups had passed about the same amount of programming courses. Furthermore, the pre-test questionnaire answered by the subjects showed that the no-UML subjects on average had slightly MORE Java experience than had the UML subjects (in total estimated lines of Java code written). We thus interpret the statement regarding “being out of practice with Java programming” as an indication of frustration over finding it difficult to get an overview of the program and not being able to solve the tasks correctly. Recall that this problem was more prevalent among no-UML subjects.

In the questionnaire and think aloud comments, no-UML subjects wrote more often than UML subjects that they did not successfully complete the tasks.

The qualitative results from the Oslo experiment suggest that there are three possible explanations for why there appears to be a significant effect of UML for the last and most complex task (Task 5) and not for the others: The effect of UML mainly seems to depend

on task complexity, but there are also UML learning effects and motivational effect at play.

Task 1 – *No Learning effects:* Most subjects updated the UML diagrams for the ATM machine after coding, and did not actively use the diagrams during coding. No further tasks were performed on the ATM.

Tasks 3 & 4 – *Low Task Complexity:* The no-UML subjects stated that UML diagrams would not have helped them for the low complexity tasks. The UML subjects stated that the UML diagrams did not help them solve these tasks.

Task 5 – *High Task Complexity & Positive Learning Effects:* Compared with the other tasks, Task 5 was quite complex, as it required a detailed understanding of the control flow in the delegated control-style of the vending machine in order to perform the change correctly [3]. The no-UML subjects report that it was very difficult to get the required overview of the code. None of the UML subjects report similar problems. The subjects had already performed two tasks on the vending machine and updated the UML diagrams twice. By updating the UML documentation, the subjects had a better opportunity to learn details about the design from updating the UML diagrams for the vending machine than the no-UML group. Five subjects reported that once they started on Task 5, they realized that the UML diagrams would be useful to understand how to solve the task.

From the above discussion, we see that qualitative analysis therefore confirms that UML is overall useful. However, that statement must be qualified as it appears to have a significant effect for the more complex tasks and only for certain people, who perhaps are more comfortable with abstraction and modeling. This is consistent with the quantitative results that show a significant improvement in correctness only for the last, most complex task. It is also clear that, due in part to the education system, coding is still perceived by some participants as the most crucial task, modeling being considered at best a secondary, somewhat wasteful activity.

The Ottawa experiment did not involve any sophisticated qualitative analysis like in Oslo. However, as discussed in Section 3.6, participants filled out questionnaires after

each lab regarding the task and, when relevant, the use of UML (Appendix B). Regarding the tasks, the results can be summarized as follows:

- The lab instructions were perceived as clear. This is an important point for the validity of our results.

- Task 6 was perceived as being more difficult than the other tasks, as expected. Task 1 was perceived as being the easiest.

With respect to UML usage, participants perceived that:

- UML diagrams were fairly easy to understand. This is important as it suggests the participants' training and modeling skills were sufficient.

- Most people spent less than 25% of the lab time understanding UML diagrams, over all tasks. For the reasons discussed above in the Oslo experiment, this result may suggest that UML diagrams were not used to a sufficient extent by a large proportion of participants. This would then tend to minimize the effects of UML we observe in our quantitative analysis.

- Results about whether UML diagrams made the system easier to understand, cleared up ambiguities, or helped complete the tasks are unclear. It is probably difficult for students to answer this question as it requires introspection in their own work.

4.5 Inconsistency

One inconsistency between the two experiments is related to Task 5. In the Oslo experiment, improvements in correctness are significant for Task 5 (the last task in Oslo), but not in the Ottawa experiment. In Ottawa, improvements in correctness were significant only for Task 6 (the last task in Ottawa). If the complexity of a task were to determine whether it is significant or not, then we would expect consistent results on the same tasks across experiments. One potential explanation could be varying capabilities between Oslo and Ottawa students. However we have seen they have a similar

education/training and there was no convincing evidence of other differences that would impact their capability. Another possibility is that learning effects play a major role and, as a result, UML effects only become visible on the last task. The question is now whether there is convincing evidence of learning effects to determine whether this is a plausible explanation. In the Oslo experiment, learning effects, if any, are confounded with the tasks and cannot be determined from the quantitative results. However, as already discussed, the *qualitative* results do indeed indicate learning effects that explain, in part, why Task 5 was significant in the Oslo experiment: when the Oslo subjects reached Task 5 (at which point they had already performed three change tasks using UML), they saw the potential benefits of UML more clearly and hence used UML more actively at that point. For Task 6 in the Ottawa experiment, we have seen in Section 4.3 that *Order* was a significant ANOVA factor for both the time (T) and design quality (Q') dependent variables, thus showing some evidence of learning effects. Such a trend is however not clearly visible on simpler tasks in the Ottawa experiment. Though this remains to be investigated, it seems plausible that the trends we observe result from the compounded effect of learning effects and task complexity.

5 VALIDITY

The construct validity of our dependent variables has already been discussed in Section 3.6. We do not think there is any serious threat in terms of internal validity as the design, through blocking and counterbalancing (in the Ottawa experiment), has attempted to remove any bias from any known extraneous factor. However, we faced a usual problem with experiments involving human subjects: even when they are properly trained, they may not follow the prescribed process and instructions, either due to fatigue or motivational problems. As a result and for reasons discussed in Section 4, we had to leave out a number of observations that were considered not valid as far as the hypotheses under investigation were concerned. Though we showed that the capability of UML and no-UML groups remained unaffected, future experiments should investigate efficient ways to capture precise data about the level of subject compliance to instructions and strategies to increase compliance.

The main weakness of such controlled experiments lies in their external validity. As it is usually the case for controlled experiments in academic, artificial settings, the participants are students and the systems being changed very small. However, those students are well-trained for the tasks: they are good programmers and have thorough knowledge of UML modeling. Though the change tasks are probably much easier than average change tasks on actual systems, we would then expect the impact of UML to be strengthened on larger tasks and systems since both our qualitative and quantitative results suggest that UML is more useful for complex tasks. This further strengthen our conjecture that our results are probably conservative in the sense that we underestimate the impact of UML.

6 CONCLUSIONS

This paper presents the results of two consecutive experiments, which have taken place in two different locations. The goal was to shed some light on the cost effectiveness of model-driven development with the UML. Because this is a very large realm of investigation, we focus here on whether models help performing quicker, better changes on existing systems. It is very common in practice to have software engineers performing changes to systems they have not developed, and maintenance consumes a large share of resources in typical software organizations. This is why we thought this was an important first question to investigate, though we realize that model-driven development can be useful in many other ways (e.g., code generation).

The results of our two experiments are mostly consistent. When only considering the time required to make code changes, UML does overall help save effort. On the other hand, when including the time necessary to modify the diagrams, no effort savings are visible. However, in terms of the functional correctness of the changes, both experiments seem to indicate UML has a significant, positive impact on the most complex tasks. In the Ottawa experiment—which investigated the design of the changes as well—UML helped to achieve higher design quality changes, which would then be expected to help future, subsequent changes. However, the above statements must be qualified. One is not likely to benefit below a certain change complexity threshold or before a certain learning curve

regarding the use of UML models for change impact analysis has been completed. Note that this comes in addition to substantial training in UML modeling. Furthermore, current tools still need substantial improvements in the way they support changes to models and consistency checking.

In terms of experimental methodology, we have found it very useful to start with a smaller experiment and dwell on qualitative analysis at first. This has allowed us to better understand what issues might come up in subsequent, larger experiments. Based on the first experiment we decided, for example, to use more complex change tasks in the second experiment. It was also clear that, to change diagrams, a complex UML tool was not required and sometimes hindered the good performance of subjects' tasks. As a result, we ended up using Visio for the second experiment, which is a much simpler diagramming tool. Furthermore, qualitative analysis is very time-consuming [14][6] and could only be used on a small-scale experiment. It was however useful to identify plausible explanations for what we observed in the second experiment. For example, we realized that the extent to which UML diagrams were used to analyze changes varied a great deal among participants.

There is a lot to be done in terms of future work. Though we do not intend to provide a detailed research plan here, it is obvious that there are many ways in which UML can be employed in the context of model-driven development. There are furthermore many profiles specializing the UML notation [27] and their impact on software engineering activities is worth investigating. The experiments we have presented here can provide practical guidelines on how to proceed to evaluate alternatives in an experimental fashion.

ACKNOWLEDGEMENTS

We thank Magne Jørgensen, Vigdis By Kampenes and Dag Sjøberg for providing valuable comments on this paper.

[6] The qualitative analysis of the Oslo experiment (conducting the interviews, transcribing the recorded interviews and performing the analysis) took several months to complete.

REFERENCES

[1] B. Anda, K. Hansen, I. Gullesen and H. K. Thorsen, "Experiences from Using a UML-Based Development Method in a Large Organization," Simula Research Laboratory, Technical Report 2005-05, 2005.

[2] B. Anda, K. Hansen, I. Gullesen and H. K. Thorsen, "Experiences from Using a UML-Based Development Method in a Large Organization," *(forthcoming) Empirical Software Engineering Journal*, 2006.

[3] E. Arisholm and D. Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Transactions of Software Engineering*, vol. 30 (8), pp. 521-534, 2004.

[4] E. Arisholm, D. Sjøberg, G. J. Carelius and Y. Lindsjørn, "A Web-Based Support Environment for Software Engineering Experiments," *Nordic Journal of Computing*, vol. 9 (4), pp. 231-247, 2002.

[5] V. Basili, F. Shull and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25 (4), pp. 456-473, 1999.

[6] K. Beck, *Extreme Programming Explained*, Addison Wesley, 2001.

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

[8] Borland: Together, 2003. www.borland.com/together.

[9] L. C. Briand, "Software Documentation: How Much is Enough?," Invited Keynote Address, *Proc. IEEE European Conference on Software Maintenance and Reengineering*, pp. 13-15, 2003.

[10] L. C. Briand, Y. Labiche, M. Di Penta and H.-D. Yan-Bondoc, "An Experimental Investigation of Formality in UML-based Development," *IEEE Transactions on Software Engineering*, vol. 31 (10), pp. 833-849, 2005.

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

[12] J. L. Devore and N. Farnum, *Applied Statistics for Engineers and Scientists*, Duxbury, 1999.

[13] R. J. Freund and W. J. Wilson, *Regression Analysis: statistical modeling of a response variable*, Academic Press, 1998.

[14] S. E. Hove and B. Anda, "Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research," *Proc. IEEE International Symposium on Software Metrics*, pp. 23-32, 2005.

[15] G. Iversen and H. Norpoth, *Analysis of Variance*, Sage Publications, second Edition, 1987.

[16] C. M. Judd and E. R. Smith, Kidder, L.H., *Research methods in social relations*, Holt, Rinehart and Winston, Inc., 6th Edition, 1991.

[17] A. Karahasanovic, B. Anda, E. Arisholm, S. E. Hove, M. Jørgensen, D. Sjøberg and R. Welland, "Collecting Feedback during Software Engineering Experiments," *Empirical Software Engineering - An International Journal*, vol. 10 (2), pp. 113-147, 2005.

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

[19] M. Kutar, C. Britton and T. Barker, "A Comparison of Empirical Study and Cognitive Dimensions Analysis in the Evaluation of UML Diagrams," *Proc. 14th Psychology of Programming Interest Group*, 2002.

[20] L. Kuzniarz, M. Staron and C. Wohlin, "An Empirical Study on Using Stereotypes to Improve Understanding of UML Models," *Proc. 12th IEEE International Workshop on Program Comprehension*, 2004.

[21] Microsoft: Visio, Version 2002, 2003. www.microsoft.com.

[22] R. H. Myers, D. C. Montgomery and G. G. Vining, *Generalized Linear Models with Applications in Engineering and the Sciences*, Wiley, 2002.

[23] OMG, "UML 1.5 Specification," Object Management Group, Complete Specification formal/03-03-01, 2003.

[24] A. N. Oppenheim, *Questionnaire Design, Interviewing and Attitude Measurement*, Pinter Publishers, 1992.

[25] T. J. Ostrand and M. J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Test," *Communications of the ACM*, vol. 31 (6), pp. 676-686, 1988.

[26] M. C. Otero and J. J. Dolado, "An empirical comparison of the dynamic modeling in OML and UML," *Journal of Systems and Software*, vol. 77 (2), pp. 91-102, 2005.

[27] T. Pender, *UML Bible*, Wiley, 2003.

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

[29] H. C. Purchase, L. Colpoys, M. McGill and D. Carrington, "UML collaboration diagram syntax: an empirical study of comprehension," *Proc. 1st International Workshop on Visualizing Software for Understanding and Analysis*, 2002.

[30] H. C. Purchase, L. Colpoys, M. McGill, D. Carrington and C. Britton, "UML class diagram syntax: an empirical study of comprehension," *Proc. Australian Symposium on Information Visualisation*, 2001.

[31] QSR: N6, 2004. http://www.qsrinternational.com/.

[32] I. Reinhartz-Berger and D. Dori, "OPM vs. UML-Experimenting with Comprehension and Construction of Web Application Models," *Empirical Software Engineering - An International Journal*, vol. 10 (1), pp. 57-79, 2005.

[33] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25 (4), pp. 557-572, 1999.

[34] E. Soloway, R. Lampert, S. Letowski, D. Littman and J. Pinto, "Designing documentation to compensate for delocalized plans," *Communications of the ACM*, vol. 31 (11), pp. 1259-1267, 1988.

[35] Telelogic: TAU, 2003. http://www.telelogic.com/products/tau.

[36] S. Tilley and S. Huang, "A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding," *Proc. 21st Annual International Conference on Systems Documentation*, pp. 184-191, 2003.

[37] J. Warmer and A. Kleppe, *The Object Constraint Language*, Addison Wesley, 2nd Edition, 2003.

[38] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and A. Wesslen, *Experimentation in Software Engineering - An Introduction*, Kluwer, 2000.

Appendix A Features concerned by changed Task 6 (Ottawa Experiment)

Ottawa Task 6: There are two similar design solutions for this task. They both involve two changes and two additions.

Solution 1:

1. Change the constructor in CoffeeMachine
2. Change the doAction() in CoffeeMachine
3. Add setRecipe() in Product
4. Add setPrice() in Product

Solution 2:

1. Change the constructor in CoffeeMachine
2. Change select() in FrontPanel
3. Add setRecipe() in Product
4. Add setPrice() in Product

Appendix B Questionnaires for the Ottawa experiment

The questionnaires that were distributed to participants are provided below. Questionnaires differed depending on whether the lab was the first one and whether it involved UML models.

Survey Questionnaire

Levels of agreement:

1 – Strongly agree 2 – Agree 3 – Not certain 4 – Disagree 5 – Strongly disagree

Questions about the task	1	2	3	4	5
1. It took me a lot of time to understand the system, before I could perform the task.	□	□	□	□	□
2. I did not have any problems to understand what the system is doing.	□	□	□	□	□
3. I had no difficulties to complete the task.	□	□	□	□	□
4. I hope there could have been more time for me to complete the task.	□	□	□	□	□
5. The task was not easy to complete.	□	□	□	□	□
6. The lab instructions were clear and easy to follow.	□	□	□	□	□
Questions about tools:	1	2	3	4	5
7. I am an expert in using Visio.	□	□	□	□	□
8. I am an expert in using JBuilder.	□	□	□	□	□
9. I have used Visio often before the Training period.	□	□	□	□	□
10. I have used JBuiler often before.	□	□	□	□	□

Figure 12. Survey Questionnaire for Lab1 (UML not provided)

Survey Questionnaire

Levels of agreement:

1 – Strongly agree 2 – Agree3 – Not certain 4 – Disagree 5 – Strongly disagree

Questions about the task	1	2	3	4	5
1. It took me a lot of time to understand the system, before I could perform the task.	□	□	□	□	□
2. I did not have any problems to understand what the system is doing.	□	□	□	□	□
3. I had no difficulties to complete the task.	□	□	□	□	□
4. I hope there could have been more time for me to complete the task.	□	□	□	□	□
5. The task was not easy to complete.	□	□	□	□	□
6. The lab instructions were clear and easy to follow.	□	□	□	□	□

Figure 13. Survey Questionnaire for Lab2, Lab3 and Lab4 (UML not provided)

Survey Questionnaire

Levels of agreement:

1 – Strongly agree 2 – Agree 3 – Not certain 4 – Disagree 5 – Strongly disagree

Questions about the task	1	2	3	4	5
1. It took me a lot of time to understand the system, before I could perform the task.	□	□	□	□	□
2. I did not have any problems to understand what the system is doing.	□	□	□	□	□
3. I had no difficulties to complete the task.	□	□	□	□	□
4. I hope there could have been more time for me to complete the task.	□	□	□	□	□
5. The task was not easy to complete.	□	□	□	□	□
6. The lab instructions were clear and easy to follow.	□	□	□	□	□

Questions about UML:	1	2	3	4	5
7. The UML diagrams in the document were easy to understand.	□	□	□	□	□
8. The UML diagrams in the document made the system easier to understand.	□	□	□	□	□
9. The UML diagrams in the document helped to clear up some ambiguities.	□	□	□	□	□
10. The UML diagrams in the document helped me to complete the task.	□	□	□	□	□
11. I hope there could have been more time for me to modify the UML diagrams.	□	□	□	□	□

12. How much time did you spend in understanding the UML diagrams during this lab?

A. <25% B. >=25% and <50% C. >=50% and <75% D: >=75%

13. How much time did you spend in modifying the UML diagrams during this lab?

A. <25% B. >=25% and <50% C. >=50% and <75% D: >=75%

Questions about tools:	1	2	3	4	5
14. I am an expert in using Visio.	□	□	□	□	□
15. I am an expert in using JBuilder.	□	□	□	□	□
16. I have used Visio often before the Training period.	□	□	□	□	□
17. I have used JBuiler often before.	□	□	□	□	□

Figure 14. Survey Questionnaire for Lab1 (UML provided)

Survey Questionnaire

Levels of agreement:

1 – Strongly agree 2 – Agree 3 – Not certain 4 – Disagree 5 – Strongly disagree

Questions about the task	1	2	3	4	5
1. It took me a lot of time to understand the system, before I could perform the task.	□	□	□	□	□
2. I did not have any problems to understand what the system is doing.	□	□	□	□	□
3. I had no difficulties to complete the task.	□	□	□	□	□
4. I hope there could have been more time for me to complete the task.	□	□	□	□	□
5. The task was not easy to complete.	□	□	□	□	□
6. The lab instructions were clear and easy to follow.	□	□	□	□	□

Questions about UML:	1	2	3	4	5
7. The UML diagrams in the document were easy to understand.	□	□	□	□	□
8. The UML diagrams in the document made the system easier to understand.	□	□	□	□	□
9. The UML diagrams in the document helped to clear up some ambiguities.	□	□	□	□	□
10. The UML diagrams in the document helped me to complete the task.	□	□	□	□	□
11. I hope there could have been more time for me to modify the UML diagrams.	□	□	□	□	□

12. How much time did you spend in understanding the UML diagrams during this lab?

A. <25% B. >=25% and <50% C. >=50% and <75% D: >=75%

13. How much time did you spend in modifying the UML diagrams during this lab?

A. <25% B. >=25% and <50% C. >=50% and <75% D: >=75%

Figure 15. Figure 4 Survey Questionnaire for Lab2, Lab3 and Lab4 (UML provided)

Appendix C Interview guide for Oslo Experiment

The information in brackets indicates which subjects were asked the given question, as follows.

[ALL] – All subjects in the experiment.
[UML] – subjects in the UML group
[NO UML] – subjects in the No-UML group

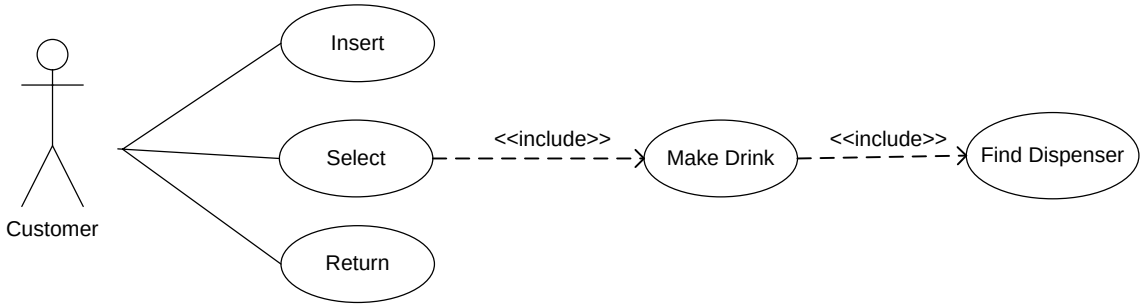
1. [ALL] What do you think of the experiment in general? (Explain briefly how it was to participate in the experiment, time pressure, stress etc)
2. [ALL] What do you think of the information that was provided (Printed information sheets, the information in SESE etc. Any confusions)?
3. [ALL] What do you think of the task formulations?
4. [ALL] Were you sometimes uncertain of what to do? Would it have helped with better instructions?
5. [ALL] How was it to use the tools TAU and SESE (Did you get problems because of the tools and what consequences did this have for you?)
6. [ALL] What do you think of the tasks' degree of difficulty?
7. [ALL] Have you solved similar tasks before?
8. [ALL] Did you work differently with the experiment tasks compared to how you normally work?
9. [ALL] Was it easy to understand the Coffee Machine program?
10. [ALL] Were any of the tasks particularly easy to solve?
    1. [ALL] Which task?
    2. [ALL] What made the task easy to solve?
    3. [ALL] How were you thinking when you made changes to the task?
11. [ALL] Were any of the tasks particularly difficult to solve?
    1. [ALL] Which task?
    2. [ALL] What made the task difficult to solve?
    3. [ALL] How were you thinking when you made changes to the task?
12. For the most difficult task [UML]:
    1. What was it like to understand the UML documentation that was provided?
    2. How useful was the UML documentation for you in order to solve the task? In which was it useful?
    3. How did you use the UML documentation in the task solving process?
    4. Have you used UML in this way before?
    5. What was the most difficult; to find out WHERE the changes should be done or HOW to do the changes?
    6. What was the most difficult; to find out WHERE/HOW to do the changes or to write the Java code?
    7. What was the most complicated; to update the UML diagrams or to write the Java code?
    8. To what degree do you think that the UML documentation helped you to understand WHERE to do the changes?

9. To what degree do you think that the UML documentation helped you understand HOW to do the changes?

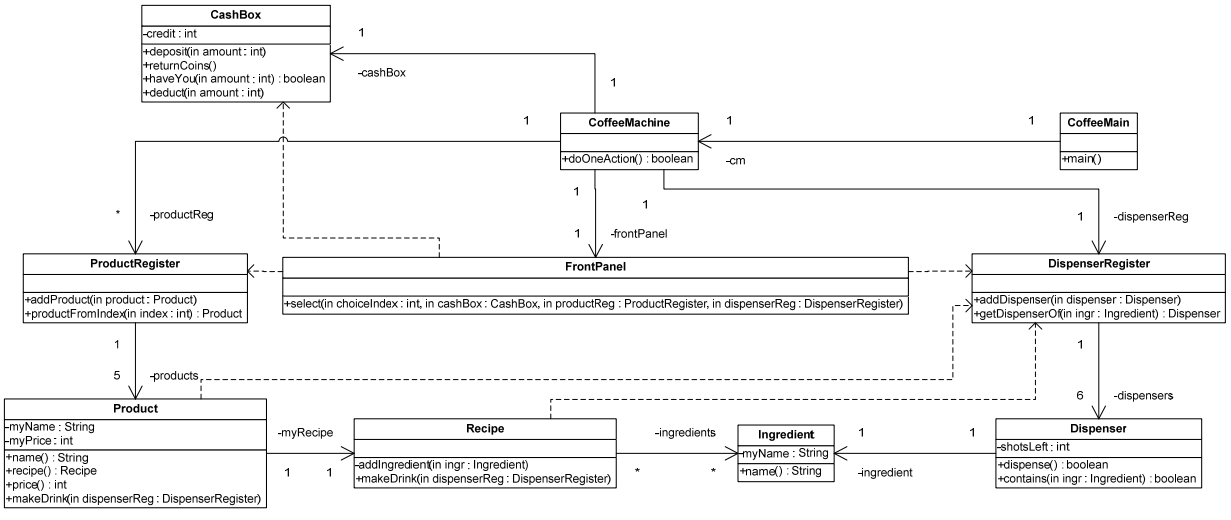
13. For the most difficult task [NOT UML]:
    1. What was the most complicated; to find out WHERE you should do the changes or HOW to make the changes?
    2. What was the most complicated; to find out WHERE/HOW to do the changes or to write the Java code?
    3. To what degree do you think UML diagrams would have made it easier to understand WHERE to do the changes (Class and sequence diagrams)?
    4. To what degree do you think UML diagrams would have made it easier to understand HOW to do the changes (Class and sequence diagrams)?
14. For the easiest task [UML]:
    1. To what degree was the UML documentation useful for you in order to solve the task? How was it useful?
    2. What was the most difficult; to find out WHERE the changes should be done or HOW to do the changes?
    3. What was the most difficult; to find out WHERE/HOW to do the changes or to write the Java code?
    4. What was the most complicated; to update the UML diagrams or to write the Java code?
    5. To what degree do you think that the UML documentation helped you to understand WHERE to do the changes?
    6. To what degree do you think that the UML documentation helped you understand HOW to do the changes?
15. For the easiest task [NOT UML]:
    1. What was the most difficult; to find out WHERE the changes should be done or HOW to do the changes?
    2. What was the most complicated; to find out WHERE/HOW to do the changes or to write the Java code?
    3. To what degree do you think UML diagrams would have made it easier to understand WHERE to do the changes (Class and sequence diagrams)?
    4. To what degree do you think UML diagrams would have made it easier to understand HOW to do the changes (Class and sequence diagrams)?
16. For those who had UML diagrams available, but did not use them [UML]:
    1. What was the reason for not using the UML documentation when you solved the tasks?
    2. Do you think that you would have made a better solution if you have used the UML diagrams?
    3. If you used the UML documentation to some extent: what information did you use, which diagrams did you look at?

Appendix D Example UML diagrams of the Coffee Machine

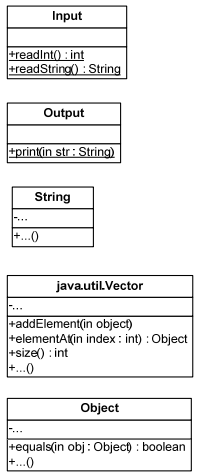
Coffee Machine Use Case Diagram



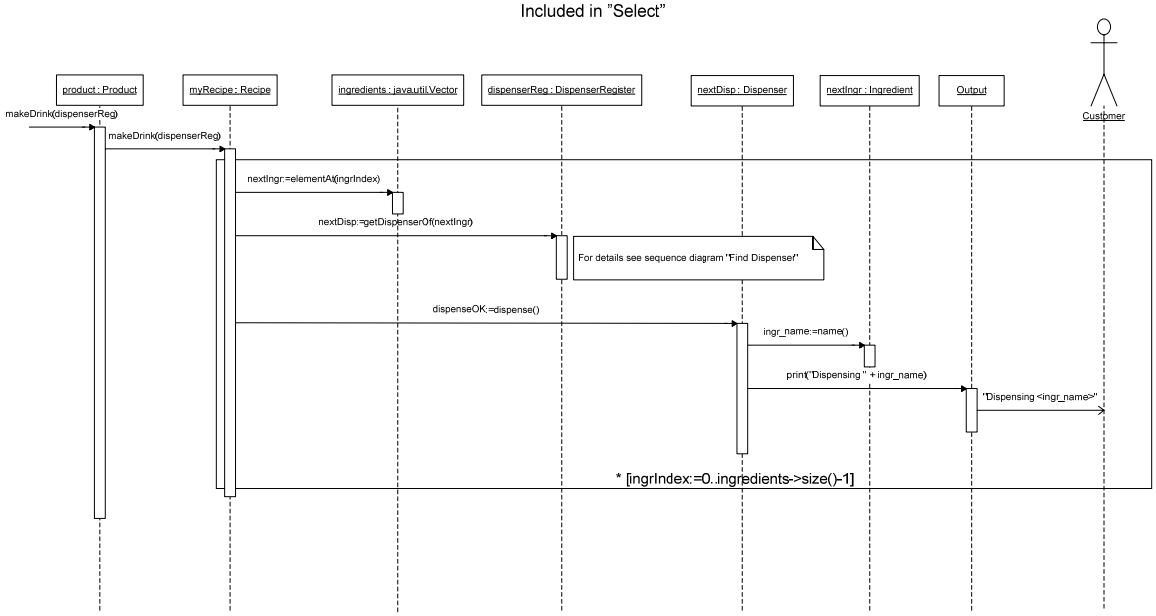
Coffee Machine Class Diagram



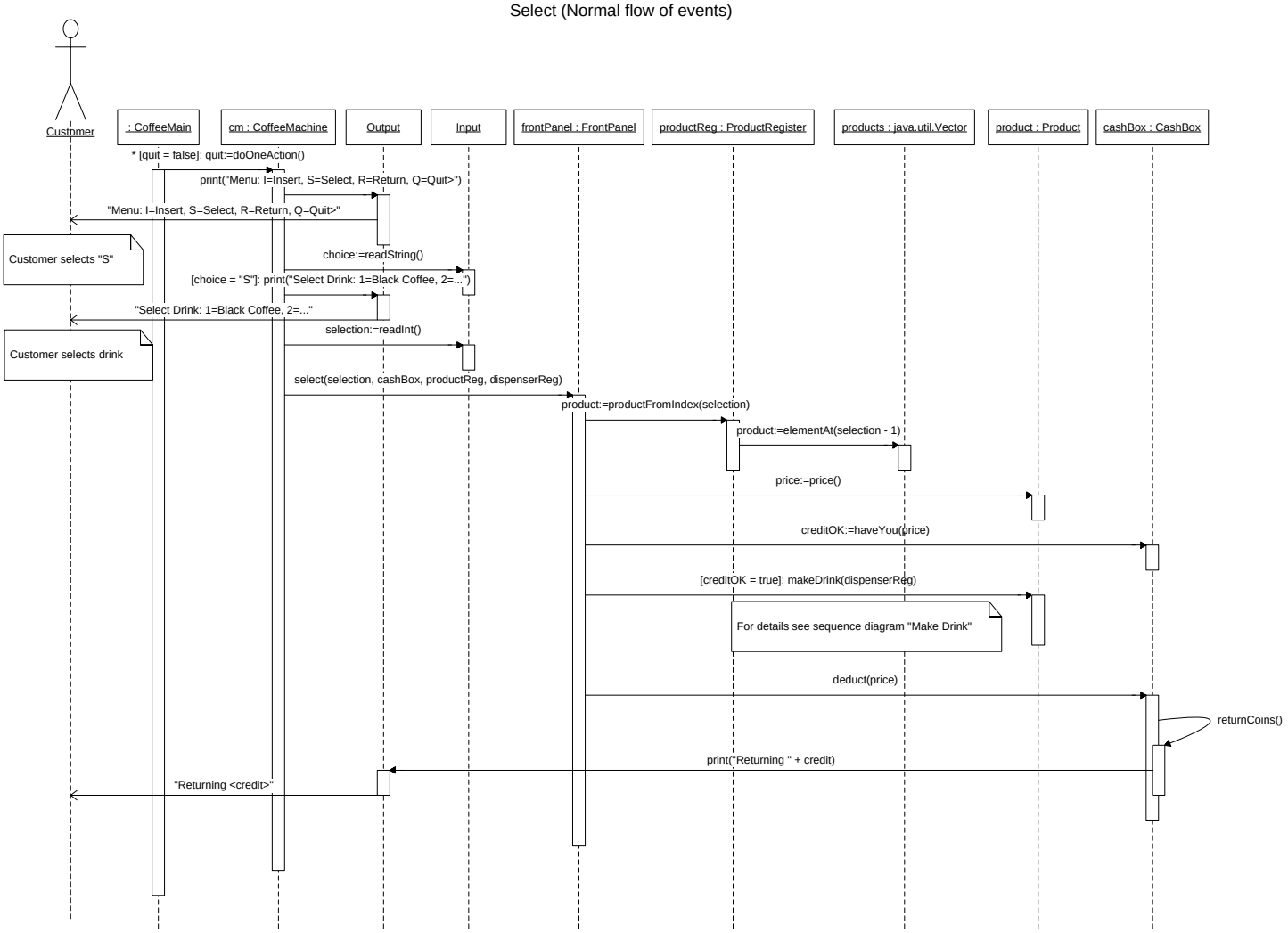
Other dependent classes



Make Drink (Normal Flow of Events)
Included in "Select"

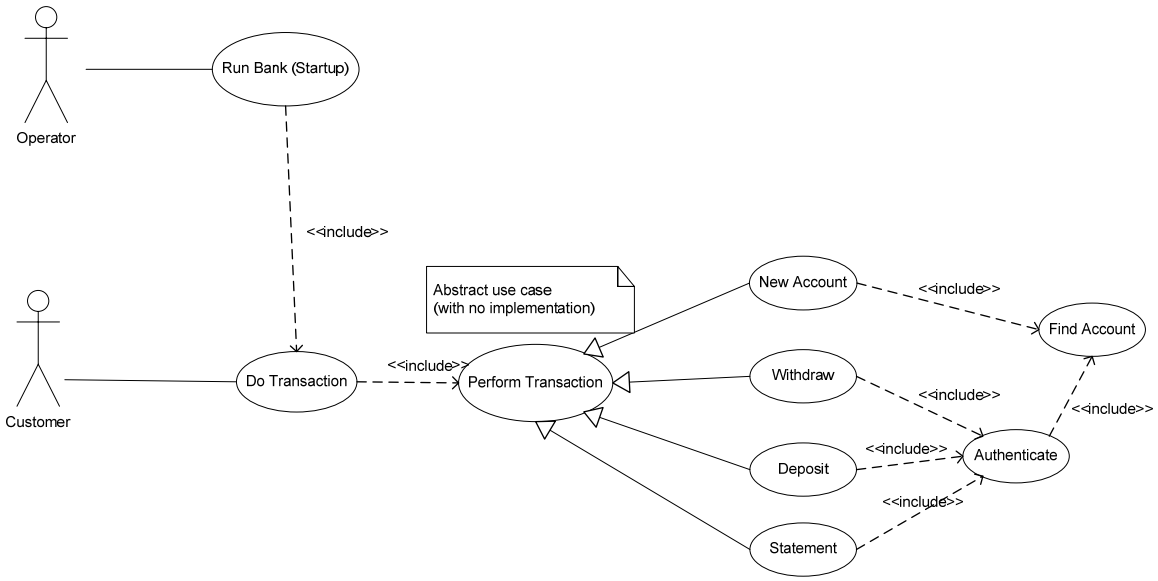


Select (Normal flow of events)

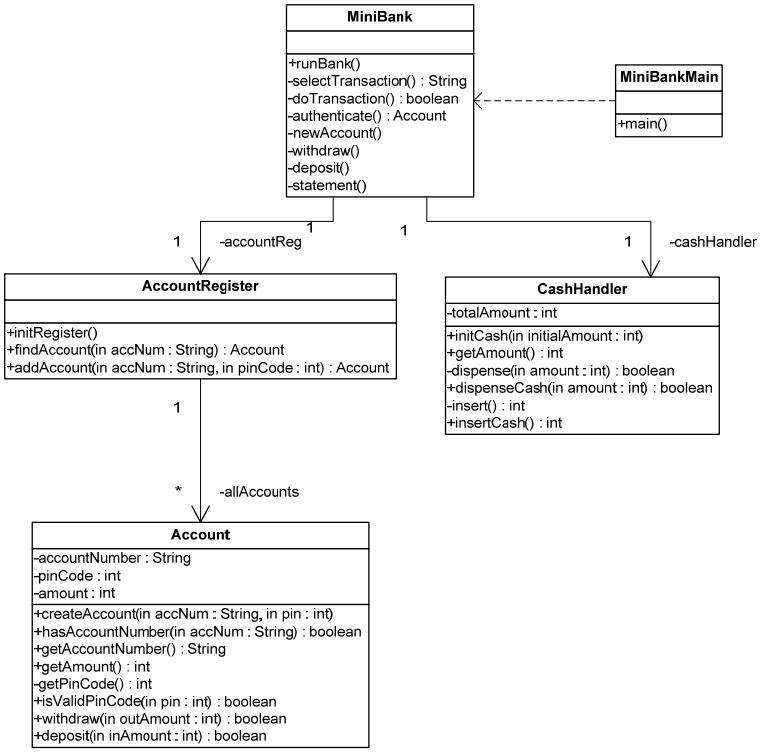


Appendix E Example UML diagrams of the ATM

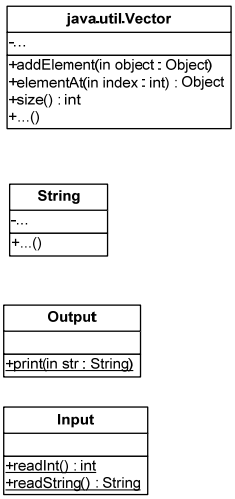
MiniBank Use Case Diagram

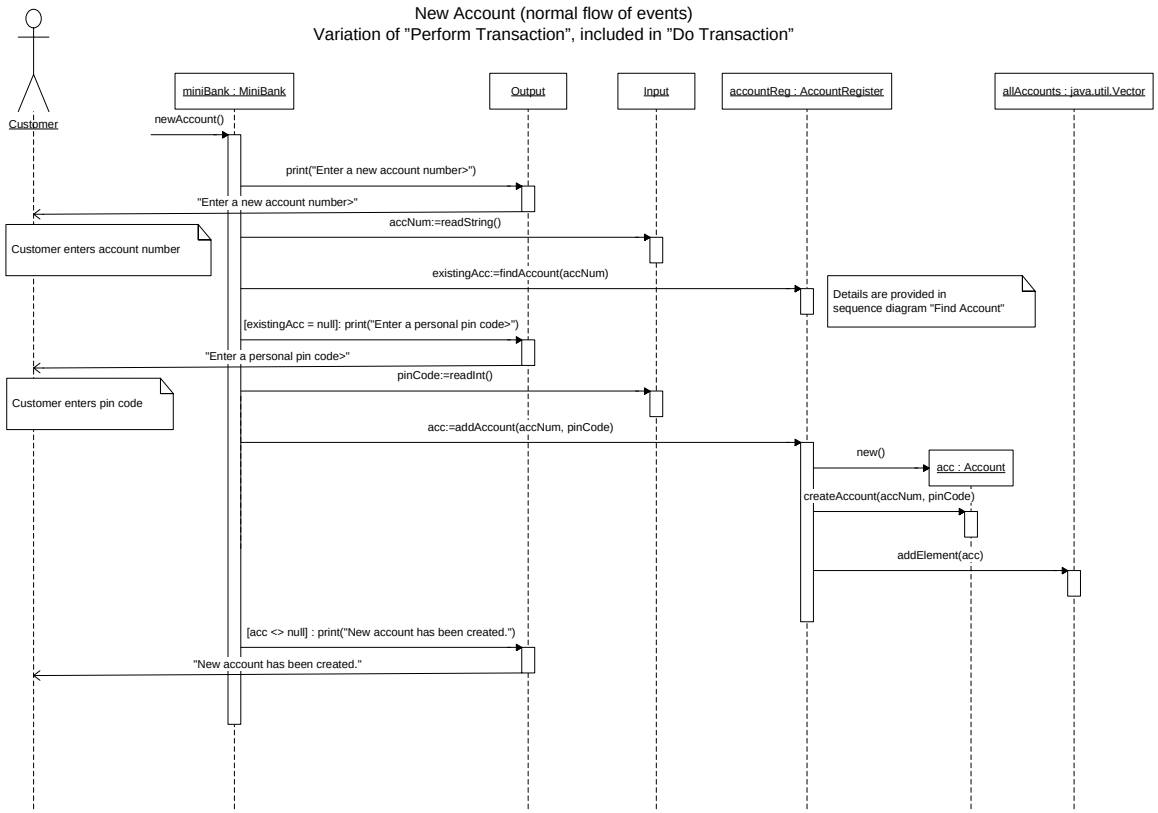


MiniBank Class Diagram



Dependent utilities





Find Account
Included in use cases Authenticate and New Account