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

Erik Arisholm, *Member*, *IEEE*, Lionel C. Briand, *Senior Member*, *IEEE*,
Siw Elisabeth Hove, and Yvan Labiche, *Member*, *IEEE*

Abstract—The Unified Modeling Language (UML) is becoming the de facto standard for software analysis and design modeling. However, there is still significant resistance to model-driven development in many software organizations because it is perceived to be expensive and not necessarily cost-effective. Hence, it is important to investigate the benefits obtained from modeling. As a first step in this direction, this paper reports on controlled experiments, spanning two locations, that 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 in the functional correctness of changes as well as the quality of their design. However, there does not seem to be any saving of time. 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.

Index Terms—Maintenance, UML, experiment.

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 [8], [12]. 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 must be maintained along with the software system it describes. The issue that then arises is what content and level of detail are required for efficient software maintenance [10]. The proponents of agile methods usually advocate keeping documentation to a minimum and focusing on test cases as a source of system requirements [7]. By contrast, proponents of model-driven development view software development as a series of modeling steps, where each step refines the models of the previous step [17]. 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 to work at higher levels of abstraction. Model-driven development is supported by the Unified Modeling Language (UML) [8], an evolving standard that is now widespread across the software industry. However, despite its 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 [7]. Hence, such practices are 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 [26] and entails the comprehension of large, complex systems under constant change.

This paper attempts to evaluate the impact of using UML documentation on the correctness and effort of performing changes. This is done on the basis of two controlled experiments that took place in two distinct geographical locations. Both involved 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 variations in results and, therefore, identify plausible and necessary conditions for UML to be effective. Our decision to address the above issues by using controlled experiments stems from the many confounding and uncontrollable factors that could blur the results in an industrial context. In a real project setting, it is usually impossible to control for factors such as ability and learning/fatigue effects and to select specific tasks to assign to individuals. 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. [6] state that "Controlled experiments can generate stronger statistical confidence in the conclusions" and Judd et al. [15]

- *E. Arisholm, L.C. Briand, and S.E. Hove are with the Simula Research Laboratory, Department of Software Engineering, Martin Linges v 17, Fornebu, PO Box 134, 1325 Lysaker, Norway. E-mail: {erika, briand, siweh}@simula.no.*
- *Y. Labiche is with the Department of Systems and Computer Engineering, Software Quality Engineering Laboratory, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada. E-mail: labiche@sce.carleton.ca.*

Manuscript received 18 Aug. 2005; revised 11 Apr. 2006; accepted 20 Apr. 2006; published online 23 June 2006.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0220-0805.

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 run for a 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." [5]. 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 controlled experiments and industrial case 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 considered. Furthermore, controlled experiments enable the assessment of 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 this paper is structured as follows: Related work is discussed in Section 2. Section 3 reports on the planning of the two controlled experiments. Results are presented in Section 4. Section 5 analyzes threats to validity and Section 6 concludes.

2 Related Work

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

An experiment was conducted to assess the qualitative efficacy of UML diagrams in aiding program understanding [34]. The subjects, whom the authors rated as UML experts, had to analyze a series of UML diagrams and complete a detailed questionnaire concerning a hypothetical software system. Results from the experiment suggest that UML's efficacy in supporting program understanding is limited by 1) unclear specifications of syntax and semantics in some of UML's more advanced features, 2) spatial layout problems, e.g., large diagrams are not easy to read, and 3) insufficient support for representing the domain knowledge required to understand a program.

A complementary work, which also investigated the impact of UML documents' content, investigated whether making UML models more precise using the Object Constraint Language (OCL) [36], which is part of the UML standard [21], helped with defect detection, comprehension, and maintenance [11]. 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 conditional upon the provision of substantial, thorough training to the experiment participants.

Other studies have compared UML to other notations in specific contexts: For example, one experiment compared the comprehension of subjects when provided with two different types of modeling notation [30]: UML models or OPM (Object-Process Methodology) models. The results suggest that OPM is better than UML for modeling the dynamics aspect of Web applications for relatively untrained users. Another experiment evaluated the comprehensibility of the dynamic models in two standard languages, UML versus OML (OPEN Modeling Language) [24]. The results suggest that specifications of dynamic behavior using OML can be understood more easily than specifications developed 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, one experiment evaluated the comprehension of sequence versus collaboration diagrams [18] and showed no significant differences in the understandability of the two alternative notations. Experimental results have also shown that the use of stereotypes may have a positive effect on model understandability [19]. Experiments reported in [27], [28] 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 investigated how access to *textual* system documentation (the requirements specification, design document, test report, and user manual) helped when performing maintenance tasks [35]. 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 pretest score) and the potential benefits of the system documentation: The most skilled maintainers benefited the most from the documentation.

The impact of the design style of an Object-Oriented (OO) system on its understandability and changeability was investigated in [3]. 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 for novices to understand. Only senior developers benefited from a delegated control style. One possible explanation of the results reported in [3] is discussed in [32]: *Delocalized plans* need to be documented explicitly 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 [32]: "*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 conducted recently in the form of a qualitative case study [1]. The participants in the case study acknowledged that, despite some difficulties (e.g., need for adequate training),

there are clear benefits to be derived 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. Yet, this is crucial if we want to see the 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 [1].

3 Experiment Planning

3.1 Experiment Definition

The UML documentation used here corresponds to what one would typically expect at the end of design, in terms of content and level of detail [12], as further described in Section 3.5.1. We first want to assess whether the provided UML documentation helps to reduce the effort required to change the source code. In other words, we are interested in whether UML documents can help to 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. It is also important to assess the functional correctness of the changes because UML documents may also help achieve better change reliability. Furthermore, because a change can be functionally correct but poorly designed, thus affecting the maintainability of the changed system, we want to assess the quality of the change's design and determine whether UML helps maintainers to achieve better designs by understanding the existing system design better.

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

3.2 Experimental Context

To answer the above research question, two distinct experiments were conducted. They differ in terms of their design, size, and focus. The first experiment took place in Oslo, Norway, and the second in Ottawa, Canada. We will refer to them as the Oslo and Ottawa experiments, respectively. The Oslo experiment involved fewer participants and, because it was the first one, it was more exploratory and therefore relied more heavily on subject interviews and qualitative analysis [31]. The Ottawa experiment involved a larger number of participants and was designed to make standard statistical analyses possible.

While, in Oslo, the participants received financial compensation for participating in the experiment, the experiment in Ottawa was part of compulsory course laboratories and the tasks were performed as practical laboratory exercises. In the latter case, we had to ensure that 1) every student would undergo the same learning experience, 2) the laboratory exercises were a valuable practical experience that supported the objectives of the course, and 3) the tasks assigned would be feasible within the scheduled laboratory hours. These considerations were paramount in selecting the proper course in which to run the experiment and for the definition of our experiment design and material. The students did not know what hypotheses were tested.

3.3 Hypotheses Formulation

Our experiment 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).

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

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 documentation 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 because we expected that using UML documentation would help people understand the system design better and, hence, provide better solutions faster. However, it is difficult to have clear expectations regarding the effect of using UML documentation on time when including time spent on diagram modifications (T') because the time taken to modify the diagrams might be greater than the expected time gains. Thus, the hypothesis on time 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: $T(UML) \geq T(no\ UML)$	H_a: $T(UML) < T(no\ UML)$
Time including diagram modifications	H_0: $T'(UML) = T'(no\ UML)$	H_a: $T'(UML) \neq T'(no\ UML)$
Correctness	H_0: $C(UML) \leq C(no\ UML)$	H_a: $C(UML) > C(no\ UML)$
Quality	H_0: $Q(UML) \leq Q(no\ UML)$	H_a: $Q(UML) > Q(no\ UML)$

TABLE 2
Summary of Competencies

	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

TABLE 3
Details of the Two Systems (for the First Task of Each System)

	ATM	Vending Machine
# of classes	7	12
# of operations	30	25
# of attributes	8	14
# of use cases	8	5
# of messages in sequence diagrams	84	54
# LOC	338	293

3.4 Selection of Subjects

The Oslo experiment involved third-year informatics students who had previously taken two courses with significant UML exposure. The Oslo students had taken different numbers and types of course, but all of them had taken at least two OO programming courses and at least two courses where UML was introduced. The Ottawa experiment involved fourth-year computer/software engineering students who 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 [33] and Visio [20]). Based on this information (summarized in Table 2), we deemed that they had the required training to perform the tasks proficiently. 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 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.

3.5 Experimental Design

3.5.1 Experimental Tasks

Both experiments involved the same two systems: 1) a simple ATM system and 2) a software system controlling a vending machine to serve hot drinks. Both systems were used in previous experiments [3] (but, of course, with other subjects) 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 [23]. This test technique was 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 explain 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 in both the UML and no-UML groups, so no bias was introduced. The UML documents provided information at a level of

TABLE 4
Systems' and Tasks' Descriptions

System	Tasks	Description	Oslo	Ottawa
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

detail that one would expect at the end of the design phase [12]: 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 described simply in English. Example UML diagrams are provided in [2].

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 further task, more difficult than the others, was therefore included for each of the ATM (Task 2) and Vending (Task 6) systems. Tasks 1 and 5 from the Oslo experiment were reused as the easiest tasks in Ottawa. In the remainder of the paper, tasks are referred to by the task names given in Table 4.

3.5.2 Time Allocation

The Oslo students had eight hours, all in one day, to complete all 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. Experience from a previous experiment [3] suggest that subjects who work fast may spend more time on the last task than they would on previous tasks (e.g., to try out alternative solutions). 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 was not considered in the analysis.

Ottawa (Carleton) students had five course laboratories of three hours each, spaced a week apart. The first session was used for exercises and additional training and each of the four tasks was then performed in subsequent laboratory sessions. There was no need for a time-sink task because students only performed one task per laboratory session and were free to leave the laboratory once their work was completed.

3.5.3 Other Factors to Be Controlled

As with any software engineering experiment, we expected a wide variation in terms of students' ability. In both experiments, we used blocking to ensure comparable skills across student groups using, and not using, UML. In both experiments, the cutoff points for the blocks were set on the basis of the actual data (collected before the experiments) to ensure that there would be an equal number of subjects in each block. In Oslo, blocking was based on the number of "passed credits" in computer science specific courses because 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 (the median value, roughly corresponding to six courses): 11 students in the low-credit (below 30 course credits) and high-credit blocks, respectively. The 11 students within each block were then assigned randomly to one of the two treatments. Two students (one from each block, both assigned to the UML treatment) did not appear for the final experiment. Despite the loss of two subjects, the mean number of credits in the resulting groups was very similar (27.0 for the UML group and 26.5 for the no-UML group). In Ottawa, we were able to 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- (the median value) 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 had already been acquired in previous courses, students received specific training for the experimental tasks at hand so that they could become 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), which required no further training, was used for UML modeling. The first laboratory session was used to refresh the students' knowledge about UML modeling.

In Oslo, all subjects performed all tasks on both systems in the same order, as shown in Table 5. In Oslo, 11 of the

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)

students were assigned to the no-UML treatment for all tasks and they all performed the same tasks in the same order. This was possible because, given that the students were paid for their involvement, there was no ethical necessity to ensure that all of them would go through the same learning experience.

In Ottawa, half the participants started with the Vending Machine, while the other half worked first on the ATM. The motivation was to ensure that neither of the two systems would benefit more from learning effects than the other. In the laboratory context of the Ottawa course, it was, of course, an absolute pedagogic requirement that all subjects be exposed to the UML treatment. Thus, 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. The experimental design for the Ottawa experiment is summarized in Table 6, which shows what task was performed, on which system, and in which order by which group. Students were assigned to each of the four groups randomly, according to the blocking procedure described above. The number of people performing tasks on the ATM first is almost equal to the number of people who worked on the Vending Machine first. 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 was used are counterbalanced. The groups are not exactly the same size due to subject loss (i.e., people missing laboratory sessions, being sick, etc.) after subjects had been assigned to the groups. However, a one-way analysis of variance shows clearly that the grade means do not differ significantly across groups.

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

3.6 Instrumentation and Measurement

In Oslo, 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, and task solutions and a comprehensive set of qualitative data pertaining to each task were collected through SESE. In addition, through SESE, the students completed 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. Note that the feedback collection method seems to be quite unobtrusive and does not appear to have introduced a bias between the groups in the Oslo experiment, as further explained in [16]. At the end of the experiment, semistructured interviews of the participants took place and were analyzed using the QSR N6 qualitative analysis tool [29], as described in Section 3.7.2.

In Ottawa, since SESE could not be used in ordinary laboratory settings, students had to download documents from a Web site and were asked to send their solutions by e-mail as soon as they were completed. In addition, after each task, students were asked to complete 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 (see [2] for detailed questionnaires).

In Oslo, test cases were devised to test the main scenario of the changed function. All task solutions were also inspected manually to assess the degree of correctness further. On the basis of the functional testing and the manual inspection, the solution to each change was graded on a six-point scale to indicate the amount of work required to fix any deviations from the prescribed functionality, as

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

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

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 based mainly 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, given that cosmetic variations in the expected output were not considered to be relevant.

In Ottawa, because there were more 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 [23]. For each change, we tried to devise the best test suites possible. However, since the changes were not that extensive, the test suites were rather small (ranging from five to 12 test cases). Only failed test cases were 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 given that 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 the assignment of class responsibilities [12].[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, 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 $m2$, correctly change $m3$ and c and have no other additions or changes would result in four correctly added/changed class elements (Q = 4) out of a maximum of four and zero incorrectly added/changed class elements (Q′ = 0). A solution that would correctly add $m1$ and change $m3$, but would omit $m2$ and the change in c, would result in two correctly added/changed class elements (Q = 2). Furthermore, if we assume that the evaluated solution implements the functionality through a substandard design that modifies one method different from c and adds one method different from $m2$ (e.g., in a different class), 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 because they offer complementary measurements of design quality, which cannot be combined meaningfully 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.

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:

1. T,
2. T',
3. a ratio scale functional correctness measure counting the number of passed test cases (denoted C' to differentiate it from C), and
4. design quality measured in two distinct and complementary ways: the number of *correctly* changed

1. In practice, there are usually a few such solutions that can minimize coupling and maximize cohesion. In our case, there were two of them for Task 6 and only one for the other tasks.

2. Note that, in general, when applying a strategy, it is necessary to identify the correct solution that is closest to each evaluated subject solution. However, to obtain comparable mesaurements, differences in model elements involved have to be considered. 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 [2]).

elements (Q) and the number of *incorrectly* changed elements (Q').

The level of significance for the hypotheses tests was set to $\alpha = 0.05$. However, the reader should bear in mind that we perform multiple tests and, in order to allow for a stricter and more conservative interpretation of the results, we provide p-values.

For 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 [13]. In addition, to reduce potential threats to validity resulting from violations of the t-test assumptions, nonparametric Wilcoxon rank sum tests are also performed [13]. 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 [13] is used to test the difference in the *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).

For the Ottawa experiment, both univariate and multivariate analyses are 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 used UML documentation for a given task with the two groups of subjects that did not use UML documentation for that same task. As in the Oslo experiment, both two sample t-tests and the nonparametric equivalent Wilcoxon rank sum tests are used to perform the univariate tests of the hypotheses. In the multivariate analyses, we performed a three-way analysis of variance (ANOVA) [13] to test the simultaneous effect of *UML, Task Order* (to assess crossover effects), *Block* (to assess ability effects), and the interactions between these factors. However, because this analysis does not yield significant new findings and the results are more difficult to interpret than univariate analysis results, we do not report them here and refer the reader to [2] for further details.

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. The questionnaire contained, among other things, 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. Seventy-six out of a total of 120 questionnaire comment fields were completed and the information stored in a database. In addition, the SESE tool polled the subjects for feedback during the experiment (see Section 3.6). Two hundred twenty-five such comments were collected and stored in a database during the experiment.

Within one week of the main phase of the experiment, semistructured interviews were conducted with 19 of the 20 subjects (one subject did not attend the interview). An interview guide with relatively open questions was prepared. The interviews were recorded on tape. Each interview lasted from 12 to 35 minutes, depending on the group to which the subjects were assigned and on how talkative the subjects were. During the interviews, a shortcoming in the interview guide was discovered. When the guide was designed, the assumption was that those assigned to the UML group would use the diagrams in order to understand the program and several questions were related to this. However, many subjects did not use the documentation in the expected way and, when this became evident, the interview guide was extended after the interviews started. The updated interview guide [2] ensured that the interviewers would not forget to ask questions about why the subjects did *not* use the UML diagrams in the way expected. The recorded interviews were transcribed carefully to increase the accuracy and comprehensiveness of the analysis.

The qualitative data from three sources (task questionnaires, feedback data, and transcribed interviews) were analyzed using QSR N6 [29], a tool for qualitative data analysis. The texts were examined, sorted, and categorized into different concepts and a tree structure with concepts and subconcepts was built on the basis of the data. The analysis attempted to assess the following:

1. how UML was used,
2. the subjects' perceptions of the costs and benefits of using it,
3. how the subjects worked, and
4. what types of problems the subjects experienced on the different tasks.

The purpose of the analysis was to better understand how 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 explain the quantitative results better, as further explained in Section 4.3.

The Ottawa experiment did not involve any sophisticated qualitative analysis of the sort that the Oslo experiment did. However, as discussed in Section 3.6, participants filled out questionnaires after each laboratory session regarding the task and, when relevant, the use of UML [2]. Standard techniques for phrasing subjective questions and designing survey questionnaires were followed [22] to avoid bias and to increase the reliability of the responses. The questions were rated on a 1-to-5 response scale. A one-way analysis of variance was used to analyze the responses.

4 Experimental Results

Recall that four hypotheses are tested with regard 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'),

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%		

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

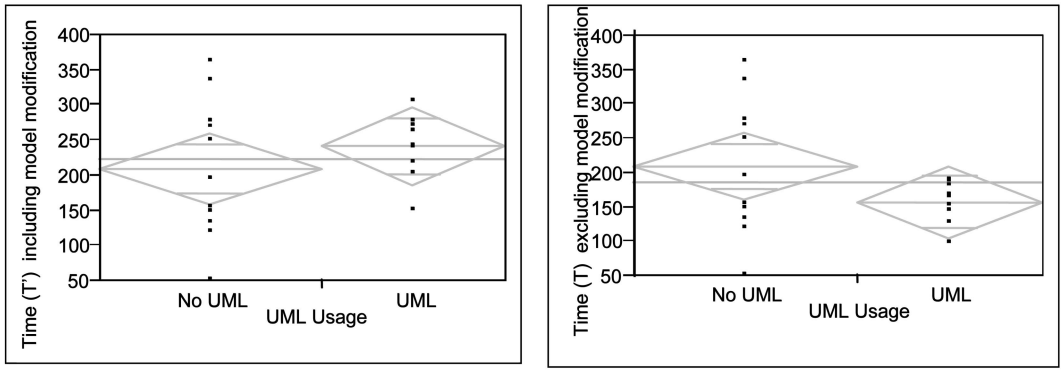
3. functional correctness (using the variable C in the Oslo experiment and C' in the Ottawa experiment), and
4. design quality (using the two complementary quality measures Q and Q').

This section first presents the overall descriptive statistics for the data that was collected in both experiments and then addresses each tested hypothesis in turn by presenting the results obtained in both Oslo and Ottawa and comparing them.

4.1 Descriptive Statistics

The descriptive statistics for 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 documentation performed 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 percent and 89 percent 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).

The descriptive statistics of the Ottawa experiment are given in Table 9. The results for the common tasks (1 and 5) are consistent with those obtained in Oslo with respect to time: Modifying models when using UML takes longer (T'). When considering only the time taken to modify code (T), the subjects who used UML seem to take more time for all tasks but Task 6. However, functional correctness (C') does not seem to improve when using UML, in contrast to what was observed in Oslo. Task 2 also shows a significant increase in time for subjects who used UML, especially when modifying models. Correctness does not seem to increase though. For Task 6, the time taken does not increase significantly, even when modifying models, whereas functional correctness (C') clearly increases (the median increases from 0/12 to 5/12). With respect to design quality (correct changes), the use of UML documentation seems to have had 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 for Tasks 1 and 6 suggest a



(a) (b)

Fig. 1. Oslo—Time to complete all four tasks (including or excluding diagram modification). (a) Including diagram modification. (b) Excluding diagram modification.

difference between the UML and no-UML groups. Note that our discussion at this point is informal and that the effect size of UML will be discussed more precisely when presenting univariate analysis results.

4.2 Univariate Analysis

The sample mean, data distribution, and 95 percent confidence interval of the mean for each dependent variable are presented in *diamond plots*, as a way to visualize the effect size of the two treatments. The line across each diamond represents the group mean and the vertical span of each diamond the 95 percent confidence interval for each group. Overlap marks are drawn below and above the means and an overlap depicts a difference that is not significant at an $\alpha = 0.05$ level. The line crossing the diagram is the entire sample mean.

4.2.1 Time to Perform the Change Tasks (T, T')

Univariate results for time with respect to the Oslo experiment are illustrated in Fig. 1a. The figure shows means diamonds of the total time spent on all four tasks when including diagram modification (T'). We can see that the UML subjects spent, on average, more time than their no-UML counterparts, but the difference is not statistically significant. If we do not consider the time spent on modifying the diagram (Fig. 1b), 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), using UML documentation does not seem to have provided an advantage in Oslo. A more detailed analysis for each task (which is not reported here) also yielded no significant difference.

In the Ottawa experiment, we encountered a specific problem: For the most complex task (Task 6), a number of solutions submitted by students did not even compile. Upon inspection, their code changes appeared to be of very poor quality and could not be easily fixed to yield a running program, let alone a functionally correct one. In any experiments with human subjects, and for a variety of reasons (e.g., fatigue, lack of motivation), some of them inevitably perform very poorly on any given task, especially in a context where there is no guarantee that all subjects will follow instructions. Because we considered time and quality through separate analyses, there being no meaningful way to combine them into one dependent variable, the issue arose as to how we could ensure we were making meaningful time comparisons. It seemed to us that the solutions should reach a certain quality threshold if time comparisons were to be meaningful. This issue was particularly important in our case, where four out of five solutions that did not compile were in the no-UML group, which had the potential to bias the results. For this reason, we decided to omit from our analysis of time any solutions that did not compile. As detailed later in this section, we checked to see whether the removal of such solutions from the analysis had introduced a new threat to the internal validity by confirming that the grade distributions were still comparable across UML and no-UML subjects.

Fig. 2 shows the time (T) distributions for each of the four tasks, which distributions clearly suggest that the subjects found successive tasks increasingly more difficult, especially Task 6.[3] In this experiment, we consider each task individually since, as we will see, different results are observed, due in part to the variation in complexity. When considering the time taken to modify models (T) (though subjects using UML documentation took, 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 $\alpha = 0.05$. However, as suggested in the Oslo experiment, the results are very different when considering the time taken to modify models (T'). For the first three tasks, the subjects using UML documentation show significantly higher time values with p-values equal to or below 0.003 (two-tailed t-test and nonparametric Wilcoxon (rank sums) test[4]).

Let us more carefully consider the results of Task 6 (Vending Machine), the most complex of the tasks, since that was the only task where using UML documentation seems to save coding time. During our monitoring of the laboratory sessions, we noticed that some people were not using the UML models with which they were provided to

3. Recall that, due to our design, this is not an ordering effect.

4. Henceforth, when a t-test result is mentioned, the reader can assume that the equivalent, nonparametric Wilcoxon (rank sums) test yields an equivalent result unless otherwise mentioned.

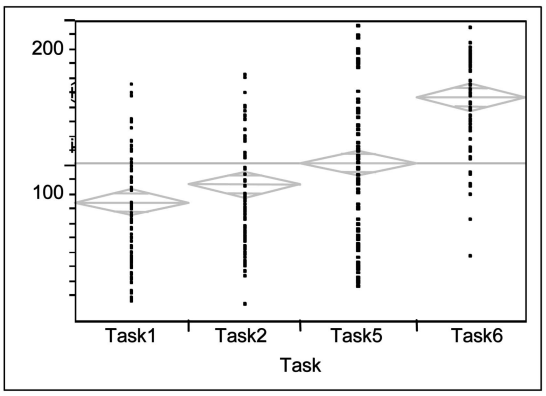


Fig. 2. Ottawa—Time (T) to complete each of the four tasks.

help them design their changes. Furthermore, some of them never returned the modified UML models. We consider these cases to be suspicious because, once the model has been used to determine the changes to be performed, it is then easy to modify the diagrams. Hence, we conclude that not returning the modified UML diagrams is an indication that these diagrams were probably not used to a great extent or possibly not at all. Further interviews with the concerned students confirmed that they mostly reverted to their old habits of relying on code. Again, as is often the case with such experiments, we could not guarantee that the prescribed process was followed precisely. As a result, the UML group improved somewhat (i.e., a reduction of 10 minutes on average), but the difference was not statistically significant (p-value = 0.19). Therefore, in order to obtain more realistic results, we decided to remove the 11 students who did not return their modified UML diagrams from the analysis and again test the significance of the difference between the UML and no-UML groups.

In order to determine whether we have introduced a new threat to internal validity by removing those students (since removing subjects may not be an entirely random process), we checked whether, after removing the solutions of these 11 students in addition to the programs that did not compile, the grade distributions were still comparable across UML and no-UML subjects. Fig. 3b confirms that this is the case because the average is nearly the same across the two groups and the variance comparable. Note that this is the grade distribution for the course where the laboratory sessions took place and is therefore a very accurate assessment of the subjects' ability with respect to the tasks they had to perform. In other words, the 16 students (12 UML, four no-UML) who were left out of the analysis were of average ability, comparable to the ability of the entire set of subjects in the UML and no-UML groups. This is an important result because the whole purpose of using blocking in our experimental design was to ensure groups of comparable ability.

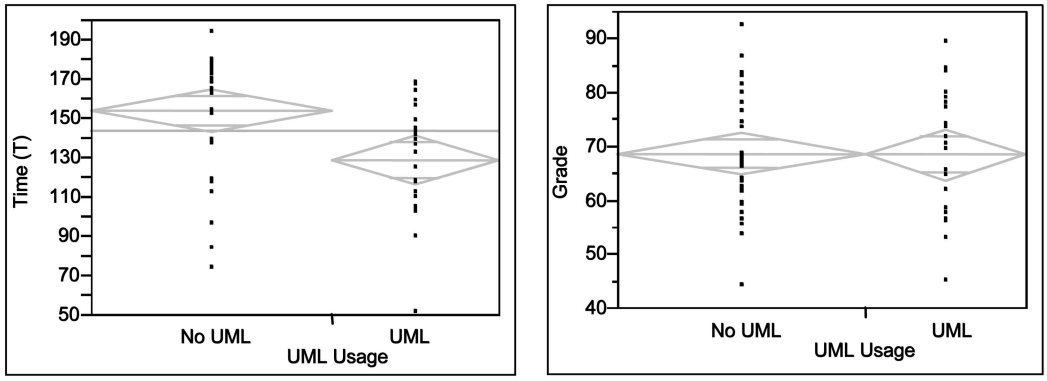
Based on the reduced data set, we can see from Fig. 3a that the average time spent by UML subjects on Task 6 is significantly lower than non-UML subjects; an average difference of over 20 minutes (on an average time of 145 minutes). A *t*-test clearly shows that the difference is statistically significant (p-value = 0.0018). In other words, when participants actually used the UML models to analyze the effect of changes, it took less time for them to change the code.

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, there is no time difference between the UML and no-UML subjects (Fig. 4). The time spent on modifying the models cancels the time saved on code changes.

4.2.2 Functional Correctness (C, C')

We first present the univariate results for correctness, C, in the Oslo experiment. Fig. 5 shows that no practically significant difference is visible for the first three tasks. However, for Task 5, the percentage of no-UML subjects who implemented the task correctly was 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 was much more difficult than the other three. Furthermore, answering the same question with a different analysis, Fig. 6 shows the percentage of correct solutions across all four tasks and we can clearly see that UML subjects performed better. However, a two-sample *t*-test yields a p-value slightly above 0.05.

In Ottawa, for reasons that were discussed in Section 3.6, we measured correctness as the number of successful test



(a) (b)

Fig. 3. Task 6—excluding cases where no modified diagrams were returned. (a) Time (T) to complete the task (excluding time to modigy UML diagrams). (b) Grade distribution.

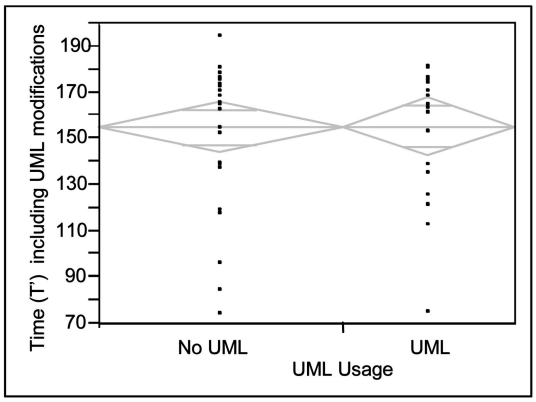


Fig. 4. Total time to complete Task 6—including time to modify the UML diagrams.

cases for each change. For the time analysis, we must decide how to handle solutions that did not compile and were therefore not testable. We think such solutions should be assigned the lowest score, that is, zero, since no test case could be run successfully and, as previously discussed, the code changes were of very poor quality. Results show, once again, that there is no significant difference between UML and no-UML subjects for all tasks but Task 6. The UML group once again shows some improvement (i.e., an additional 1.5 successful test cases, on average), but the difference is not statistically significant (p-value = 0.2).

For the same reasons as in the time analysis, we perform a second analysis in which changes where no modified model was provided are omitted. As illustrated in Fig. 7, the resulting data 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).

4.2.3 Design Quality (Q, Q')

Another dependent variable we consider in the Ottawa experiment is the quality of the design of the solutions. As described in Section 3.6, design quality was first measured as a percentage of correctly changed, deleted, and added features (Q). When comparing the design quality of UML and no-UML solutions, once again, a significant difference is observed only for Task 6. As an example, for Task 6, there are two alternative, acceptable solutions [2]. For each provided solution, the authors went through the code and compared it with either of the two alternative solutions, whichever was the closer. Fig. 8a shows that UML solutions were designed better in terms of correctly changed elements (the average difference is 0.50 on a five-point scale). This difference is statistically significant (p-value = 0.032) when running a one-tailed t-test. Furthermore, when considering the number of incorrectly changed elements (Q'), which was the other relevant measure of design quality, Fig. 8b shows a statistically significant difference in this case as well (average difference = 1.34, p-value = 0.0043). Even if we look at it in a binary way by analyzing proportions, we see that, when not using UML documentation, roughly 41 percent of the solutions contain at least one incorrect change, whereas no incorrect changes were found in any of the solutions that used UML documentation. A likelihood ratio chi-square test for proportions shows this is statistically significant (p-value < 0.001).

4.2.4 Summary

UML documentation does not seem to provide an advantage when considering the additional time needed to modify models. Even disregarding the modification of models, subjects who used UML documentation did 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 synchronization. Certain tools, such as Together from Borland [9], have already started to do so and we expect this technology to improve in the future.

In terms of correctness, despite the fact that the two experiments use different correctness measurement (see Section 3.6), both experiments show that, for the most complex task, the subjects who used UML documentation performed significantly better than those who did not. There are two main plausible ways to interpret these results. 1) Due to learning effects, subjects really benefited from using UML documentation after they had performed the first three tasks (but bear in mind that this is only true for half the subjects in the Ottawa experiment). 2) Another possibility is that they only benefited for the most complex task, i.e., Task 5 and Task 6 for the Oslo and Ottawa experiments, respectively. However, recall that the most

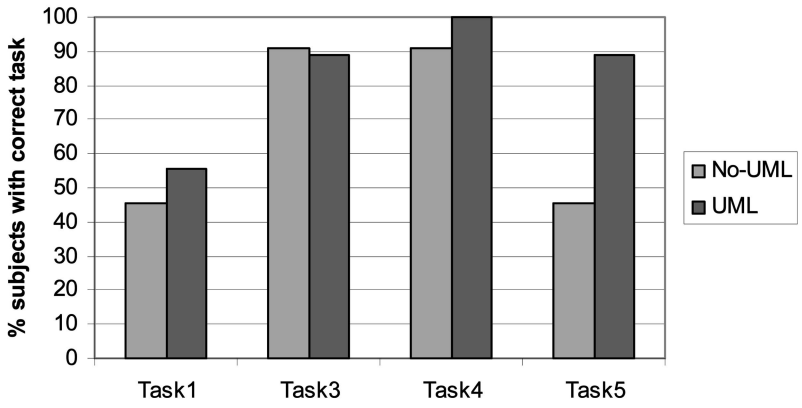


Fig. 5. Oslo—Percent of subjects with correct solutions.

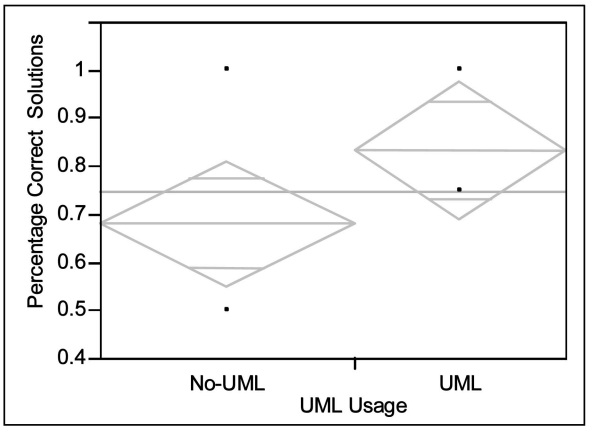


Fig. 6. Oslo—Percentage of correct solutions.

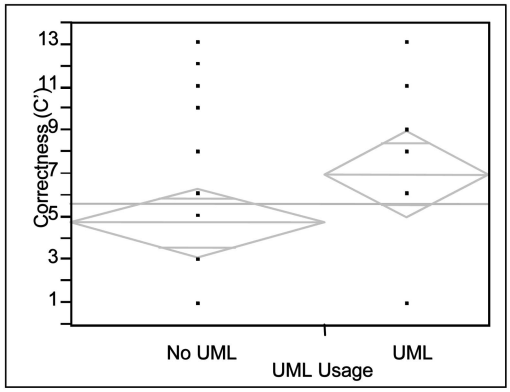


Fig. 7. Correctness for Task 6.

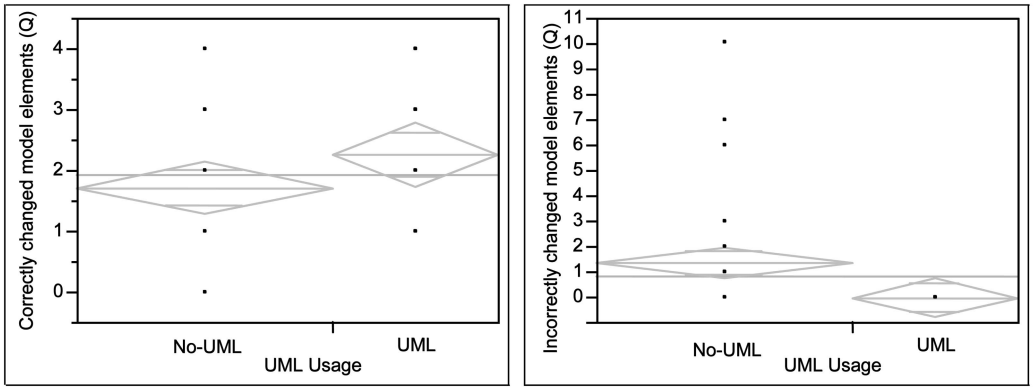
complex Oslo task was the second-most complex Ottawa task. However, if this second hypothesis is correct, the issue arises as to why we did not obtain consistent results in the Ottawa experiment for Task 5. It is therefore plausible that what we observed is the compounded result of high task complexity and learning effects.

A similar comment can be made regarding our measures of the design quality of subjects' solutions. The subjects who used UML developed superior designs for the most complex task (Task 6) of the Ottawa experiment. This is probably due to the fact that using UML models helped the subjects to understand the existing design better and, hence, helped them to make appropriate design decisions for the most complex task. For the simpler task, using UML documentation is probably not necessary.

4.3 Qualitative Analysis

For the Oslo experiment, based on interviews, the questionnaire comments, and the "think aloud" feedback comments (Section 3.7.2), the qualitative analysis yielded the results summarized in Table 10 (one subject in the UML group did not participate in the interview). The results suggest that the extent to which UML documentation was used, and its impact, varied among the UML subjects. The experiment required that all subjects update the diagrams before they moved on to the next task. However, the extent to which they used the UML models to identify change locations prior to performing code modifications varied greatly among subjects. Seven of the nine subjects assigned to the UML group updated the UML documentation after they had completed the coding. Only two subjects used the UML documents actively during development, e.g., to identify change locations. Some complained that the diagrams were very large and difficult to update, particularly the sequence diagrams. Several expressed dissatisfaction with TAU, despite having received extensive training in that tool. This led us to use a simpler tool (Visio) in the Ottawa experiment. The general consensus among the subjects was that the diagrams were more useful for more complex tasks and were often superfluous on easier tasks. This observation in the Oslo experiment led us to be suspicious of solutions without modified diagrams in the Ottawa experiment and led to our decision to leave such observations out of the analysis. Since the education system in both locations is 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 relevant for the Ottawa experiment as well.

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:



(a) (b)

Fig. 8. Design quality for Task 6.

TABLE 10
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 complete 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 completing tasks (4/9)	• UML would have been useful for locating the code changes (8/10) • UML would have been helpful with deciding how to make 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 completion (6/10)
Problems	• It was bothersome and difficult to update the UML diagrams (3/9) • Dissatisfaction with Tau UML (6/9) • Java-related problems at the beginning of the experiment (2/4***) • Out of practice with Java programming (2/9)	• Java-related problems at the beginning of the experiment (4/6***) • Out of practice with Java programming (7/10) • Problems with getting an overview of the program (10**)

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

- They were more used to looking at the code than diagrams. This is not surprising, given typical computer science curricula where people start learning how to code long before they see their first model.
- They thought 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 noncoding task as wasteful.
- The tasks were relatively small; therefore, it was sufficient to rely on the code.

Nevertheless, half of the UML subjects considered viewing UML documentation to be useful for obtaining an overview of the code, although the perceived benefits varied widely. 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 were solving the more difficult tasks, in particular Vending Task 5. Several thought that UML diagrams would have helped them to locate code changes and some claimed that it would have helped them to decide how to solve the tasks. Furthermore, some mentioned that UML documentation would have helped them to devise better solutions. UML documentation was considered to be most useful for performing complex tasks.

Another striking result that we believe is related to the above is that seven out of 10 no-UML subjects said that they were out of practice with programming/Java, whereas only two out of nine UML subjects did. However, due to the randomized blocking scheme, the subjects in the two groups had passed about the same amount of programming courses. Furthermore, the pretest questionnaire completed by the subjects showed that the no-UML subjects had, on average, 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 perform 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 complete the tasks successfully.

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 seems to depend mainly on task complexity, but there are also UML learning effects and a 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 and 4—*Low Task Complexity*. The no-UML subjects stated that UML diagrams would not have helped them for low complexity tasks. The UML subjects stated that the UML diagrams did not help them to complete these tasks.

Task 5—*High Task Complexity and 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 reported that it was very difficult to get the required overview of the code. None of the UML subjects reported 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 began Task 5, they realized that the UML diagrams would be useful for understanding how to perform the task.

From the above discussion, it is evident that qualitative analysis confirms that using UML documentation is useful, overall, with the qualification that 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, which 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 of the sophisticated qualitative analysis that the Oslo experiment involved. However, as discussed in Section 3.6, participants completed questionnaires after each laboratory session regarding the task and, when relevant, the use of UML [2]. Regarding the tasks, the results can be summarized as follows:

- The laboratory 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 thought that:

- UML diagrams were fairly easy to understand. This is important because it suggests that the participants' training and modeling skills were sufficient.
- Most people spent less than 25 percent of the time in laboratory sessions understanding UML diagrams, over all tasks. For the reasons given above in discussion of 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 tend to minimize the effects of UML that we observed in our quantitative analysis.
- Results about whether UML diagrams made the system easier to understand, cleared up ambiguities, or helped in completing the tasks are unclear. It is probably difficult for students to answer this question because it requires reflection upon their own work.

4.4 Inconsistency

One inconsistency between the two experiments is related to Task 5. In the Oslo experiment, but not in the Ottawa experiment, improvements in correctness were significant for Task 5 (the last task in Oslo). 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, we would expect consistent results on the same tasks across experiments. One potential explanation could be varying capabilities between Oslo and Ottawa students. However, as we have seen, they had similar education/training and there was no convincing evidence of other differences that would affect their capability. Another possibility is that learning effects played a major role and that, as a result, the effects of using UML only became visible on the last task. The issue is, therefore, whether there is convincing evidence of learning effects so that it can be determined 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 using UML more clearly and, hence, used it more actively at that point. For Task 6 in the Ottawa experiment, the multivariate analysis reported in [2] showed 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. However, such a trend was not clearly visible on simpler tasks in the Ottawa experiment. Though this remains to be investigated, it seems plausible that the trends we observed resulted from the compounded effects 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 because the experiment was designed, through blocking and counterbalancing (in the Ottawa experiment), to remove any bias from any known extraneous factor. However, we faced a common problem that arises 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 of this, and for reasons discussed in Section 4.2, we had to omit 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 the UML and no-UML groups remained unaffected, future experiments should investigate efficient ways of capturing precise data about the level of subject compliance with instructions and of developing strategies to increase compliance.

The main weakness of such controlled experiments lies in their external validity. As is usually the case for controlled experiments in academic, artificial settings, the participants are students and the systems being changed are very small. However, those students are well-trained for the tasks: They are good programmers and have a thorough knowledge of UML modeling. Though the change tasks are probably much easier than average change tasks on actual systems, we would expect the effect of using 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 strengthens our conjecture that our results are probably conservative, in the sense that we have underestimated the benefits of using UML documentation.

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 UML. Because this is a very large area of investigation, we focused on whether models help software engineers to make quicker and better changes to existing systems. It is very common, in practice, to have software engineers making changes to systems they have not developed and maintenance consumes a large portion of the resources in typical software organizations. This is why we thought that 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 considering only the time required to make code changes, using UML documentation does help to save effort overall. On the other hand, when including the time necessary to modify the diagrams, no savings in effort are visible. However, in terms of the functional correctness of the changes, both experiments seem to indicate using UML has a significant, positive impact on the most complex tasks. In the Ottawa experiment, which also investigated the design of the changes, using UML helped to achieve changes with superior design quality, which would then be expected to facilitate future, subsequent changes. However, the above statements apply only with qualifications. Benefits are not likely to be derived if the tasks to be performed lie below a certain level of complexity or if software engineers have not reached a certain level of skill regarding the use of UML models for analyzing the effects of changes, *in addition* to having received substantial training in UML modeling. Furthermore, current tools still need substantial improvements in the way they support changes to models and the checking of consistency.

With respect to experimental methodology, we have found it very useful to start with a smaller experiment and focus on qualitative analysis at first. This has allowed us to better understand what issues might arise 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 subjects' performance of their tasks. As a result, we used Visio for the second experiment, which is a much simpler tool for making diagrams. Furthermore, qualitative analysis is very time-consuming [14][5] 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 much 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 in the UML notation [25] and their impact on software engineering activities is worth investigating. The experiments we have presented here can provide practical guidelines on how to evaluate alternatives experimentally.

Acknowledgments

The authors thank Magne Jørgensen, Vigdis By Kampenes, and Dag Sjøberg and the anonymous reviewers for providing valuable comments on this paper. Thanks to Chris Wright for proofreading. The authors are also grateful to the students who took part in these experiments and hope they enjoyed the pedagogical experience.

References

[1] B. Anda, K. Hansen, I. Gullesen, and H.K. Thorsen, "Experiences from Using a UML-Based Development Method in a Large Organization," *Empirical Software Eng. J.,* to appear.
[2] E. Arisholm, L.C. Briand, S.E. Hove, and Y. Labiche, "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation," Technical Report 2005-14, Simula Research Laboratory, 2005.
[3] E. Arisholm and D. Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Trans. Software Eng.,* vol. 30, no. 8, pp. 521-534, Aug. 2004.
[4] E. Arisholm, D. Sjøberg, G.J. Carelius, and Y. Lindsjørn, "A Web-Based Support Environment for Software Engineering Experiments," *Nordic J. Computing,* vol. 9, no. 4, pp. 231-247, 2002.
[5] V. Basili, "The Role of Experimentation in Software Engineering: Past, Current, and Future," *Proc. IEEE Int'l Conf. Software Eng.,* pp. 442-449, 1996.
[6] V. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Trans. Software Eng.,* vol. 25, no. 4, pp. 456-473, July/Aug. 1999.
[7] K. Beck, *Extreme Programming Explained.* Addison Wesley, 2001.
[8] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide.* Addison Wesley, 1999.
[9] Borland: Together, 2003, www.borland.com/together.
[10] L.C. Briand, "Software Documentation: How Much Is Enough?" *Proc. IEEE European Conf. Software Maintenance and Reng.,* Invited Keynote Address, pp. 13-15, 2003.

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

[11] L.C. Briand, Y. Labiche, M. Di Penta, and H.-D. Yan-Bondoc, "An Experimental Investigation of Formality in UML-Based Development," *IEEE Trans. Software Eng.,* vol. 31, no. 10, pp. 833-849, Oct. 2005.
[12] B. Bruegge and A.H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java,* second ed. Prentice Hall, 2004.
[13] J.L. Devore and N. Farnum, *Applied Statistics for Engineers and Scientists.* Duxbury, 1999.
[14] S.E. Hove and B. Anda, "Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research," *Proc. IEEE Int'l Symp. Software Metrics,* pp. 23-32, 2005.
[15] C.M. Judd, E.R. Smith, and L.H. Kidder, *Research Methods in Social Relations,* sixth ed. Holt, Rinehart, and Winston, 1991.
[16] A. Karahasanovic, B. Anda, E. Arisholm, S.E. Hove, M. Jørgensen, D. Sjøberg, and R. Welland, "Collecting Feedback during Software Engineering Experiments," *Empirical Software Eng. —An Int'l J.,* vol. 10, no. 2, pp. 113-147, 2005.
[17] A. Kleppe, J. Warmer, and W. Bast, *MDA Explained—The Model Driven Architecture: Practice and Promise.* Addison-Wesley, 2003.
[18] 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.
[19] L. Kuzniarz, M. Staron, and C. Wohlin, "An Empirical Study on Using Stereotypes to Improve Understanding of UML Models," *Proc. 12th IEEE Intl Workshop Program Comprehension,* 2004.
[20] Microsoft: Visio, Version 2002, 2003, www.microsoft.com.
[21] OMG, "UML 1.5 Specification," Object Management Group, Complete Specification formal/03-03-01, 2003.
[22] A.N. Oppenheim, *Questionnaire Design, Interviewing and Attitude Measurement.* Pinter Publishers, 1992.
[23] T.J. Ostrand and M.J. Balcer, "The Category-Partition Method for Specifying and Generating Functional Test," *Comm. ACM,* vol. 31, no. 6, pp. 676-686, 1988.
[24] M.C. Otero and J.J. Dolado, "An Empirical Comparison of the Dynamic Modeling in OML and UML," *J. Systems and Software,* vol. 77, no. 2, pp. 91-102, 2005.
[25] T. Pender, *UML Bible.* Wiley, 2003.
[26] R.S. Pressman, *Software Engineering—A Practitioner's Approach,* seventh ed. McGraw Hill, 2005.
[27] H.C. Purchase, L. Colpoys, M. McGill, and D. Carrington, "UML Collaboration Diagram Syntax: An Empirical Study of Comprehension," *Proc. First Int'l Workshop Visualizing Software for Understanding and Analysis,* 2002.
[28] H.C. Purchase, L. Colpoys, M. McGill, D. Carrington, and C. Britton, "UML Class Diagram Syntax: An Empirical Study of Comprehension," *Proc. Australian Symp. Information Visualisation,* 2001.
[29] QSR: N6, 2004, http://www.qsrinternational.com/.
[30] I. Reinhartz-Berger and D. Dori, "OPM vs. UML-Experimenting with Comprehension and Construction of Web Application Models," *Empirical Software Engineering—An Int'l J.,* vol. 10, no. 1, pp. 57-79, 2005.
[31] C.B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Trans. Software Eng.,* vol. 25, no. 4, pp. 557-572, July/Aug. 1999.
[32] E. Soloway, R. Lampert, S. Letowski, D. Littman, and J. Pinto, "Designing Documentation to Compensate for Delocalized Plans," *Comm. ACM,* vol. 31, no. 11, pp. 1259-1267, 1988.
[33] Telelogic: TAU, 2003, http://www.telelogic.com/products/tau.
[34] 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 Ann. Int'l Conf. Systems Documentation,* pp. 184-191, 2003.
[35] E. Tryggeseth, "Report from an Experiment: Impact of Documentation on Maintenance," *Empirical Software Eng.: An Int'l J.,* vol. 2, no. 2, pp. 201-207, 1997.
[36] J. Warmer and A. Kleppe, *The Object Constraint Language,* second ed. Addison Wesley, 2003.

Erik Arisholm received the MSc degree in electrical engineering from the University of Toronto and the PhD degree in computer science from the University of Oslo. He has seven years industry experience in Canada and Norway as a lead engineer and design manager. He is now a researcher in the Department of Software Engineering, Simula Research Laboratory and an associate professor in the Department of Informatics, the University of Oslo. His main research interests are object-oriented analysis and design and design quality measurement. He is a member of the IEEE.

Lionel C. Briand received the PhD degree in computer science, with high honors, from the University of Paris XI, France. He is currently a visiting professor at the Simula Research Laboratories, Oslo, Norway. He is on a sabbatical leave from the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is a full professor and has been granted the Canada Research Chair in Software Quality Engineering. Before that, he was the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany. He also worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA Goddard Space Flight Center, CSC, and the University of Maryland. He has been on the program, steering, or organization committees of many international, IEEE, and ACM conferences. He is the co-editor-in-chief of *Empirical Software Engineering* (Springer) and is a member of the editorial board of *Systems and Software Modeling* (Springer). He was on the board of the *IEEE Transactions on Software Engineering* from 2000 to 2004. His research interests include: model-driven development, testing and quality assurance, and empirical software engineering. He is a senior member of the IEEE.

Siw Elisabeth Hove has been working at the Simula Research Laboratory for two and a half year as a research assistant. Her research interest is qualitative research methods, in particular semistructured interviews. At the moment, she is working as a consultant in the Norwegian IT industry.

Yvan Labiche received the BSc degree in computer systems engineering from the graduate school of engineering, CUST (Centre Universitaire des Science et Techniques, Clermont-Ferrand), France. He completed a Master of fundamental computer science and production systems in 1995 (Université Blaise Pascal, Clermont Ferrand, France). While working on the PhD degree in software engineering, completed in 2000 at LAAS/CNRS in Toulouse, France, he worked with Aerospatiale Matra Airbus (now EADS Airbus) on the definition of testing strategies for safety-critical, on-board software, developed using object-oriented technologies. In January 2001, he joined the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada, as an assistant professor. His research interests include: object-oriented analysis and design, software testing in the context of object-oriented development, and technology evaluation. He is a member of the IEEE.