An Experimental Investigation of Formality in UML-Based Development

Lionel C. Briand, *Senior Member*, *IEEE*, Yvan Labiche, *Member*, *IEEE Computer Society*, Massimiliano Di Penta, *Member*, *IEEE*, and Han (Daphne) Yan-Bondoc

Abstract—The Object Constraint Language (OCL) was introduced as part of the Unified Modeling Language (UML). Its main purpose is to make UML models more precise and unambiguous by providing a constraint language describing constraints that the UML diagrams alone do not convey, including class invariants, operation contracts, and statechart guard conditions. There is an ongoing debate regarding the usefulness of using OCL in UML-based development, questioning whether the additional effort and formality is worth the benefit. It is argued that natural language may be sufficient, and using OCL may not bring any tangible benefits. This debate is in fact similar to the discussion about the effectiveness of formal methods in software engineering, but in a much more specific context. This paper presents the results of two controlled experiments that investigate the impact of using OCL on three software engineering activities using UML analysis models: detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes. The results show that, 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. But, this result is however conditioned on providing substantial, thorough training to the experiment participants.

Index Terms—Comprehension of software models, software engineering experimentation, UML, OCL.

1 Introduction

The Object Constraint Language (OCL) [28] was proposed as a way to bring additional precision to system analysis and/or design models described using the Unified Modeling Language (UML) [4]. OCL is currently part of the UML standard [20]. However, a number of authors recommend against using OCL [12], [18][1] or leave it out of their proposed methodologies [13]. Some recommend using OCL only during low-level design [8]. Furthermore, very few organizations using UML make use of OCL.

Though it is clear that OCL brings precision to UML models and offers a number of potential benefits [7], [10], [17], the question comes down to assessing whether the additional effort and formality associated with OCL bring any tangible benefits in practice. This question is akin to the old, on-going debate in software engineering regarding the degree of formality required in the early phases of software development to develop high-quality software.

There is anecdotal evidence that using some kind of formal approach in the early phases of software development brings benefits [14] (e.g., less overall effort, fewer faults produced), but there are few rigorous empirical evaluations of those benefits and the factors that influence them. However, three empirical studies have been reported which are of particular interest in our context. In the first one, Pfleeger and Hatton performed a case study on the use of Finite State Machines, VDM and Communicating Sequential Processes [25]. The main results suggest that using formal specifications can lead to code that is relatively simple and easy to test. The authors, however, also report that their results are far from conclusive due to a lack of control and incomplete data as the study is a postmortem analysis. Sobel and Clarkson performed a quasi experiment[2] on the use of a formal method based on first-order logic [27]. The main conclusion is that the use of formal analysis (pre and postconditions in first-order logic) during software development produces better, less erroneous programs. However, in [3], it is pointed out that this experiment presents severe threats to validity (e.g., lack of randomization, lack of control) and therefore needs to be replicated with a different design. In particular, the experiment did not exclude the possibility that the benefits may be due simply to the nature of the people who volunteered to learn the formal treatment. In the third study, Satpathy et al. [26] performed an industrial case study aiming to compare the use of UML and B for the specification of a real-time event driven system. One of their conclusions is that the specification task benefits from both UML and the additional formality brought by B, and they even suggest combining the two by adding B statements to

1. "Unless there is a compelling practical reason to require people to learn and use the OCL, keep things simple and use natural language," see [18]. "Unless you have readers [e.g., designers] who are comfortable with predicate calculus, I'd suggest using natural language," see [12].

2. A quasi-experiment is an experiment which is unable to control potential factors that may influence the results, for instance, because of a lack of randomization [29].

- *L.C. Briand and Y. Labiche are with the Software Quality Engineering Laboratory, Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S5B6, Canada. E-mail: {briand, labiche}@sce.carleton.ca.*
- *M. Di Penta is with the RCOST—Research Centre on Software Technology, Department of Engineering, University of Sannio, Piazza Roma, I-82100 Benevento, Italy. E-mail: dipenta@unisannio.it.*
- *H. Yan Bondoc is with Nortel, 3500 Carling Avenue, Ottawa, ON K2H 8E9, Canada. E-mail: dyan@nortel.com.*

Manuscript received 2 Mar. 2005; revised 22 June 2005; accepted 22 July 2005; published online 3 Nov. 2005.
Recommended for acceptance by Harman, Korel, and Linos.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0050-0305.

express certain properties that cannot be readily expressed in UML. They also mention that the use of OCL should be investigated in future work.

This paper reports on a series of two controlled experiments performed in a university setting. Their purpose was to understand whether the OCL has an impact on three software engineering activities, that is, 1) the detection of defects in UML models, 2) the comprehension of a system's functionality, behavior, and structure based on UML models, and 3) the maintenance of UML documents, with a particular focus on change impact analysis. The subjects are fourth year computer and software engineering students who have been carefully trained in UML-based software development over several courses. As further discussed below, the choice of performing a controlled experiment was made to control other extraneous factors which could have affected the results (e.g., students' ability) and to ensure that the subjects were all adequately trained (i.e., have sufficient knowledge of UML analysis and OCL).

Though effort was a constant here—each student spent roughly the same time to perform the activities—our goal was to determine whether OCL would make a practically significant difference. Results showed that the use of OCL, combined with UML, offers significant benefits, in terms of defect detection, comprehension, and maintenance of UML analysis documents. However, the significant benefits are obtained only after a certain learning curve is overcome.

This paper is structured as follows: Section 2 provides details of the design of the experiment, including the experiment definition, context selection, hypotheses formulation and instrumentation. Section 3 reports on the results and provides plausible interpretations. Section 4 discusses threats to validity. Conclusions in terms of practical significance of the results and future work are reported in Section 5.

2 Experiment Planning

Two experiment trials have successively been conducted and are referred to as *Experiment I* and *Experiment II*, respectively, in the rest of this paper. Sections 2.1 to 2.7 describe *Experiment I*, following the template provided in [29], a well-known textbook on software engineering experimentation. *Experiment II* is a replication of *Experiment I*. Although some changes in the design of the experiment are performed, most of the descriptions in Sections 2.1 to 2.7 apply also to *Experiment II*, and differences between *Experiment I* and *Experiment II* are contrasted in Section 2.8.

2.1 Experiment Definition

The purpose of the experiment is to evaluate the impact of using OCL during object-oriented analysis [8] (i.e., when class diagrams, statecharts, sequence diagrams, etc., are first produced), on the effectiveness of three typical software engineering activities:[3]

1. **Understanding the analysis document**. Different people have to understand analysis models at different stages of software development: During the analysis phase, different analysts have to produce a common, coherent, set of UML diagrams. During system design, once analysis has been completed [8], designers (likely to be different from the analysts) have to work from the analysis document. If OCL expressions help engineers to understand the analysis, then using OCL may save the effort required to clarify issues with analysts. Furthermore, errors discovered during analysis, and/or misunderstanding of the analysis during the design phase (possibly leading to wrong design choices), may be very expensive if these are not caught at an early stage of development.
2. **Modifying the analysis document**. During software development, it is common for system requirements to change and for faults to be corrected. Each type of change may require that the UML model be changed and a small change may lead to many other related changes. For example, a change in a class may lead to changing the state definitions of this class and other classes or the pre/postconditions of operations. If the presence of OCL expressions helps to point out the impact of a change[4] easily, correctly, and quickly, then the effort of devising OCL expressions would be more justified.
3. **Detecting defects in the analysis document**. Defects are likely to exist in a UML model. They could be due for instance to a misunderstanding of the system requirements or miscommunication among team members. Usually, models are subject to inspection, to improve their correctness and completeness. If the OCL expressions were shown to help engineers to detect more defects in the models before they are being used in subsequent steps, the effort of defining OCL expressions during analysis would be further justified.

According to standard experiment design terminology [29], our experiment has one independent variable or factor, denoted as *Method*, with two treatments: *OCL* and *no_OCL*, corresponding to whether OCL expressions are used in UML analysis models. The experiment has three *dependent variables*, namely, *Comprehension (C)*, *Maintainability (M)*, and *Defect Detection (D)*, which are defined to measure the effectiveness of the three above-mentioned software engineering activities. Determining the effect of the factor *Method* on these three dependent variables is the objective of the experiments reported here.

2.2 Context Selection

The context of the experiment is a fourth year Computer and Software Engineering course at Carleton University, Ottawa, Canada. The students have all been trained in UML-based object-oriented software development in at least three previous courses, with an increasing focus on software modeling. The focus of this last course is to make them use what was learnt in the context of a well-defined development process, including analysis and design phases [8].

3. Many other software engineering activities could have been considered. But those three were selected as they correspond to typical expectations from people using UML models. The higher level of abstraction of models should help them understand a system, modify it, and the model should help avoid introducing defects in early stages of development.

4. Both in the UML diagrams and the code.

```
Title::setForSale()
Description:     Makes the title (i.e., all the copies for the title) available
                 for sale. This will be possible if the title is not reserved,
                 and if no copy is rented or held by a customer.
Parameter-List:
Return-List:
Pre:             self.state = #ForRent
                 and  not self.reservation->exists(r: Reservation|
                        r.state=#Pending  or  r.state=#Outstanding)
                 and  not self.copy->exists(c: Copy |
                        c.state=#OnHold  or  c.state=#Rented)
Post:            state = #ForSale
                 and  self.copy->forAll(c:Copy | c.state = #ForSale)
```

Fig. 1. Example of operation described using a textual description and OCL contracts.

Experimental tasks are performed on two system analysis documents, related to two real-world applications, namely, a Cab Distribution (CD) system and a Video Store (VS) system. Each system was selected because it has an adequate level of complexity and represents a familiar application domain.[5] Each analysis document (CD or VS) is approximately 40 pages long, and they are somewhat comparable in terms of complexity and size. Each analysis document entails: a partial use case diagram along with use case descriptions following standard templates [8], sequence diagrams for a subset of the use cases in the use case diagram, a couple of statecharts along with a textual description of their states and transitions, a class diagram including every model element (e.g., operation) used in sequence diagrams and statecharts, and a data dictionary reporting on classes, attributes, operations, and class associations. For the versions of the analysis documents containing OCL constraints, OCL was used to document the following aspects of the models, which are typically documented in natural language: operation preconditions and postconditions and class invariants (in the data dictionary), guard conditions in statecharts, and path conditions in sequence diagrams. Fig. 1 shows an example of operation described using both natural language and OCL preconditions and postconditions. More details on each analysis document are reported in [5].

Since our objective is to assess the impact of OCL, it is particularly important to note that both OCL and non-OCL documents contain all the information needed to perform any of the *Defect Detection*, *Comprehension*, or *Maintenance* tasks. Where present, the OCL contracts will provide details that may otherwise be spread throughout the diagrams. They also provide additional precision, thus avoiding ambiguities such as the ones that are inherent to textual descriptions. For example, the OCL precondition and postcondition in Fig. 1 show exactly the meaning, in terms of class attributes, of the phrases "*no copy is rented or held*" and "*all copies for the title are available for sale*." Similarly, the OCL expression fills the precision gap between the phrase "*if the title is not reserved*" and the fact that a title is considered reserved when there is at least one pending or outstanding reservation for it (i.e., second and third lines of the precondition). It is worth noting that a subject working on documents with no OCL expressions can infer the same information from other parts of the document (e.g., other parts of the data dictionary, statecharts).

5. Note that these models have little or no inheritance. The role of inheritance in this context will need investigation. However, our conjecture is that inheritance will increase even further the need for using OCL contracts, for instance, to check the conformance of hierarchies to the Liskov Substitution Principle [19].

2.3 Hypothesis Formulation

The following null hypothesis (H_0) is formulated for testing the effect of *Method* on each *dependent variable*: There is no difference in the subjects' *Comprehension (C)*, *Maintenance (M)*, and *Defect Detection (D)* effectiveness while working on UML analysis documents using or not using OCL. The alternative hypothesis (H_a) is that using OCL improves effectiveness for all three *dependent variables*. The formal definitions are provided in Table 1: The rows present the three *dependent variables*; the columns summarize the corresponding null and the alternative hypotheses.

2.4 Selection of Subjects

The subjects are the students registered in the last, most advanced software engineering course in their four-year Bachelor program, as it is desired that the subjects possess a reasonable level of technical maturity and knowledge for UML-based, object-oriented software development. The students are familiar with concepts such as pre and postconditions, state invariants, and class invariants. There is no need to screen students as variations in ability are also present in industrial settings and all levels of ability should therefore be represented. As most of these students will become professionally registered engineers, they are a representative sample of the population of new graduates entering the software engineering profession.

The experiment was part of a series of compulsory laboratory exercises. As students are concurrently attending lectures and doing assignments as they go through the labs, we expect some level of learning effects that will have to be accounted for in our analysis. The students were told that they were not graded on performance but that they were expected to perform their tasks individually in a professional manner to obtain the points assigned to the laboratory. They were aware of the pedagogical purpose of the exercises—that is to experience modeling tasks in the presence or absence of OCL—but did not know the exact hypotheses tested.

TABLE 1
Experiment Hypotheses

		Null hypothesis	Alternative hypothesis
Dependent variable	Comprehension	H_0: C(OCL) = C(no_OCL)	H_a: C(OCL) > C(no_OCL)
	Maintenance	H_0: M(OCL) = M(no_OCL)	H_a: M(OCL) > M(no_OCL)
	Defect Detection	H_0: D(OCL) = D(no_OCL)	H_a: D(OCL) > D(no_OCL)

2.5 Experiment Design

This section carefully presents the design choices we made to optimize the validity of the experiment. Section 2.5.1 defines the tasks performed and time allocated. Section 2.5.2 discusses other undesirable factors, whose effects may confound with the effect of using OCL. Section 2.5.3 describes the experiment design and how the extraneous factors are controlled.

2.5.1 Experimental Tasks and Time Allocation

Three tasks are defined to study the dependent variables. The *Defect Detection* task is formulated to study dependent variable *Defect Detection*: Thirty defects are seeded in each analysis document, and the subjects are asked to find them all during a period of 150 minutes without a priori knowing the number of seeded defects. The subjects' performance for this task is measured as the percentage of seeded defects detected. The *Comprehension* task is formulated to study dependent variable *Comprehension*: The subjects have to answer to 20 multiple choice questions for each system, during a period of 90 minutes. The subjects' performance for this task is measured as the percentage of correctly answered comprehension questions. The *Maintenance* task is formulated to study dependent variable *Maintenance*: A list of system changes is prescribed for each system, affecting 71 and 60 model elements in the CD and VS systems, respectively. The subjects are asked to identify all the model elements affected by the system changes during a period of 60 minutes, without knowing how many model elements are indeed affected. The subjects' performance for this task is measured as the percentage of affected model elements correctly identified. A complementary measure that was considered but not presented here due to space constraints is the total number of wrongly identified model elements: We did not observe significant differences. Note that the experiment does not deal with the efficiency or cost-effectiveness of the tasks as time was rather constrained and we were expecting a ceiling effect in terms of effort spent on each task.

As the subjects perform the three tasks on the same system, *Defect Detection* has to be performed first: The subjects are first given an erroneous document to perform the *Defect Detection* task, and then a correct document to answer comprehension and maintenance questions. Moreover, since the *Comprehension* task is believed to be simpler than the *Maintenance* task[6] (the latter task also requires understanding the systems), *Comprehension* is performed before *Maintenance*. Due to time limitations imposed on each laboratory, the *Defect Detection* task is completed in one laboratory, while the *Comprehension* and *Maintenance* tasks are completed in a subsequent laboratory, with the *Comprehension* task being performed before the *Maintenance* task. More time is allocated to the *Comprehension* task (90 minutes) than to the *Maintenance* task (60 minutes), as it is assumed that a thorough understanding of each system would help the subjects in performing the subsequent *Maintenance* task.

6. Results (Section 3) confirm this intuition.

2.5.2 Other Factors to Be Controlled

Method is the only factor of interest in this experiment. However, other factors may also impact the subjects' performance in an undesirable way and their effect may be confounded with *Method*'s effect. Extraneous factors need to be controlled so that only the effect due to *Method*, if there is any, is discernable.

1. **Academic background**. Some subjects may come from a program, such as electrical engineering, where the software development training received may not have been as intensive compared to that received by subjects coming from a software/computer engineering program. On the other hand, students in the software engineering program might be more receptive to the tasks than students in the other two programs.
2. **OCL experience**. Some subjects may have learned OCL prior to this course, while others may not have.
3. **Ability**. The subjects may have different levels of understanding of UML, OCL, and object-oriented system analysis.
4. **Order of Method**. As indicated below, subjects apply one treatment (OCL, no-OCL) for a system and then the other treatment for the other system (e.g., CD with OCL and then VS without OCL). The order in which the subjects are exposed to OCL may impact the observable effect of OCL as learning effects may be confounded with *Method*.
5. **System**. The subjects' performance may depend on their comprehension of the application domain of each system. Further, system complexity may also have a confounding effect with *Method*. For example, the effect of OCL may be more or less discernable in a more complex system.

2.5.3 Experiment Design

Students were assigned to one of four groups and the tasks they performed over four laboratories (three hours each), as well as the order of the tasks, are summarized in Table 2. The rows represent the four subsequent labs students went

TABLE 2
Experiment Design

	Task	Group 1	Group 2	Group 3	Group 4
Lab 1 (week 1)	Defect Detection	CD (OCL)	CD (no_OCL)	VS (OCL)	VS (no_OCL)
Lab 2 (week 2)	Comprehension, Maintenance	CD (no_OCL)	CD (OCL)	VS (no_OCL)	VS (OCL)
Lab 3 (week 3)	Defect Detection	VS (no_OCL)	VS (OCL)	CD (no_OCL)	CD (OCL)
Lab 4 (week 4)	Comprehension, Maintenance	VS (OCL)	VS (no_OCL)	CD (OCL)	CD (no_OCL)

through (each lab being a week apart) and the three tasks they performed, whereas the columns are the four groups of students. The table cells show, for each group and lab, which system the students worked on and whether they had OCL constraints for it.

This experiment involves only one factor whose effect is of interest to us: whether or not OCL is used in Analysis documents. We could have used a simple, completely randomized one-factor design, but we wanted to account for individual differences in such a way as to increase the statistical power of our analysis and ensure the design would not create any systematic bias in our results [15]. One way to achieve these goals is to adopt a randomized block design [29] and to account for subjects' ability during data analysis (Section 2.7). Another advantage of proceeding this way is that we can study the interaction between subjects' ability and OCL.

Students were therefore grouped into two blocks (hereby referred to as *High Ability* and *Low Ability*) according to whether or not they had obtained a minimum grade of B- in the previous course on UML and OCL. Admittedly, other ways to capture ability and form blocks could have been considered, but our focus here was on their UML modeling capability and, as instructors of their previous courses on this topic, we felt it was the best alternative. Subjects were grouped also according to whether they learned OCL in a prerequisite course or in the course (the material and number of hours of lectures used were the same), and according to the undergraduate engineering program they were registered in. Each of the four student groups was then randomly assigned subjects from blocks in nearly identical proportions. It was also important for us, in order to facilitate the data analysis, that each of the four groups be of approximately the same size (9 or 10 in this case, for a total of 38 students—see [5] for the exact figures).

For each task, each subject worked individually on each of the two systems, using UML+OCL in one case and only UML in the other. Students were also prevented from collaborating and individual work could easily be monitored as tasks were performed in a laboratory setting.

According to the design shown in Table 2, students performed each task twice, each time on a different system and with a different treatment (*OCL* or *no_OCL*). For the sake of brievity in our discussions, the first time a task is performed (Lab 1 for *Defect Detection*, Lab 2 for *Comprehension* and *Maintenance*) will be referred to as the *first attempt*, while the second time (Lab 3 for *Defect Detection*, Lab 4 for *Comprehension* and *Maintenance*) as the *second attempt*.

The rationale for making subjects work on both systems was 1) to maximize the number of data points (observations) so as to increase statistical power, 2) to ensure that the differences in systems' complexity would not bias the results (though we tried to select systems of "similar" complexity), and 3) to give the opportunity to every student to learn the same material. However, when asking experimental subjects to perform several times similar tasks (e.g., *Defect Detection*) and use the same techniques (e.g., OCL contracts), we are subject to learning or fatigue effects [15] and we need to ensure they do not confuse the results. This was done in two ways:

1. First, two of the groups used OCL the first time they performed the task whereas the other two groups did so the second time around, thus ensuring learning effects (regarding the tasks, UML, and OCL) would not be confounded with OCL effects. As described in Section 2.7, our data analysis procedure will however account for learning effects and quantify them.
2. Second, we used four groups of participants instead of two in order to avoid ordering effects (e.g., learning or fatigue effects) being confounded with system effects. For example, if all participants in group A would have used first the VS system with OCL (and then the CD system without OCL) while all participants in group B would have started with non-OCL documents on the CD system (and then VS with OCL), we could have observed that OCL helps more on the VS system as people in group B had more time to further mature their knowledge of UML and OCL. We would have then been unable to assess learning effects or system effects as they would be entirely confounded.

Another result of our design choices is that each group uses each of the two systems twice in a row, sometimes using OCL first or second. One may therefore ask whether this is a threat to the validity of the results. First, it is important to observe that, when a system is used twice in a row, it is to perform different tasks and the data collected will be part of separate analyses (i.e., for different

dependent variables). So, if there is a system learning effect, say for Group 1 from Lab 1 to Lab 2, this learning will result in a better *Comprehension* and *Maintenance* performance. However, that learning effect will affect *all* four groups and we can still pool together the four groups' data for analysis purposes within each of the labs. Furthermore, we expect that system learning effect to be weak as labs take place a week apart and students do not have access to documents between labs. Therefore, we also do not expect them to remember OCL expressions from one week to the other, and certainly not the level of detail where the performance of non-OCL groups would get closer to that of OCL groups. However unlikely, even in the situation where people would remember OCL expressions from one week to the other (e.g., Group 1 from Lab 1 to Lab 2), this would result in decreasing the impact of OCL but could not create an inflated difference between OCL and non-OCL performance scores.

All the students were monitored by the experimenters as the tasks are performed. They were aware that our goal was to evaluate the impact of using OCL as well as to improve their practical training in UML and OCL, but they were not aware of the exact hypotheses tested or what particular dependent variables were of interest (Section 2.4). The subjects were not allowed to communicate with each other during the laboratories, and were required to hand in all experimental materials at the end of each laboratory.

Two additional, related issues that can be raised from the chosen experiment design are the possible exchange of information among students between labs and the possibility they might have kept working on the tasks outside the laboratory environment, thus invalidating our assumption that the effort spent on the tasks is nearly constant. This behavior was prevented in several ways. Students were not aware of the detailed plan for the labs and, therefore, did not know they would have to perform the same tasks as their fellow students from other groups at a later point. Second, as mentioned above, students did not have access to the lab material between labs, whether the system descriptions or questionnaires. As a result any collaboration or continuation of the work between labs was unlikely and difficult.

2.6 Instrumentation

The materials used in this experiment comprise: UML analysis documents with or without OCL constraints, and with or without seeded defects (Section 2.6.1), questionnaires for the *Comprehension* and *Maintenance* tasks (Section 2.6.2), and a postlab survey questionnaire (Section 2.6.3). Additionally, to verify if the blocks are appropriate (Section 2.5.3), a prelab survey questionnaire is administered to obtain information about the background of the subjects, e.g., their previous UML experience, the undergraduate program to which they belong [5].

Each seeded defect, comprehension question, and maintenance question was carefully selected according to a number of criteria: It had to cover different aspects/parts of the systems, to the largest extent possible, however excluding implementation details; it should not be trivial or nearly impossible to answer based on the available information; and it could be answered with or without OCL. In other words, each OCL and non-OCL document contains, in different ways, relevant information to answer questions. If we had introduced question that could be answered only with OCL expressions, then the experiment would have no point as the results would then become obvious. Such questions are, anyway, difficult to find, and are expected to be rare in practice. Last but not least, the questions had to be relevant, that is had to concern information of interest. In other words, we had to be convinced that those would be questions that software engineers could ask about the system. For example, maintenance changes had to be plausible changes. Standard techniques of phrasing subjective questions and designing survey questionnaires were followed [23], so as to avoid bias and to increase reliability of responses.

2.6.1 The UML Analysis Documents

Two systems are used in this experiment: the Cab Distribution system (CD), and the Video Store system (VS). There are four versions of each of the system documents: with or without OCL expressions, and with or without defects. Each document was carefully reviewed by the authors. Though smaller than industrial analysis/specification documents, each analysis document is far from being trivial.

To perform the *Defect Detection* task, 30 defects were seeded in the correct UML document versions, containing or not containing OCL expressions. The subjects were asked to identify all the seeded defects and submit the findings via e-mail. To make the measurement objective and repeatable, the subjects' performance is measured as the percentage of seeded defects correctly identified. Defects, typically encountered in UML documents, were seeded in class diagrams, data dictionaries, sequence diagrams, and statecharts. Example of these defects are missing/wrong associations in class diagrams, inconsistencies between diagrams and data dictionary, missing/wrong sequences in sequence diagrams, missing/wrong states, transitions guard conditions, or actions in statecharts. The distribution of the seeded defects is similar in the two systems [5].

2.6.2 Questionnaires for the Comprehension and the Maintenance Tasks

During the *Comprehension* and the *Maintenance* tasks, the correct analysis documents with or without OCL are provided to the subjects, and *Comprehension* and *Maintenance* questionnaires are distributed.

The *Comprehension* questionnaire contains 20 multiple-choice questions, each one having a unique correct answer. Its purpose is to quickly evaluate, in a repeatable and objective way, the extent to which a subject understands the system internal logic and functionality, which is measured as the percentage of all questions correctly answered. Fig. 2 shows an example of a multiple choice question used for the VS system. Note how the OCL postcondition in Fig. 1 provides an answer to the comprehension question in Fig. 2. When a `Title` is set for sale, its state transitions to the state `OnSale`, and the same happens to all copies of that `Title` (i.e., "B" is the correct answer). This, however, does not require changing the `SalePrice` attribute of class `Title`

Comprehension Question	Maintenance Question
Which attribute(s) in which class(es) are modified (and how are they modified) by a call to setForSale on a Title? A. The attribute onSale of Copy class is set to "True". B. The Title state is set to ForSale and all copies are transited to state ForSale. C. The Title state is set to ForSale. The attribute salePrice is set to an amount. D. All copies are transited to the state ForSale. The attribute salePrice is set to an amount. E. Other answer: _____	**The video store company decides to limit the maximum number of unfulfilled reservations that a member can perform in a day. Each member may have a different limit.** 1. Which class(es) must be modified? 2. Which modifications (addition, change of attributes, operations, and/or operations' contracts) must be made in this (these) class(es)?

Fig. 2. Example of Comprehension and Maintenance questions.

since that attribute has already been set when a `Title` was created (to specify fees for lost copies). To answer the same question without OCL, it is necessary, in addition to reading relevant text descriptions, to analyze the textual descriptions of `Copy` and `Title` attributes in the data dictionary. We can then determine that `SalePrice` is initialized when the corresponding `Title` instance is created. In general, it may also be needed to look at the description of class statecharts and interactions of relevant objects in sequence diagrams.

To investigate on how OCL could have helped maintainers, we identified six and five prescribed changes for CD and VS, respectively, and the *Maintenance* questionnaires focus on impact analysis of these changes. Seventy-one (respectively, 60) model elements are impacted in the CD (respectively, VS) system by the changes. The subjects are asked to list model elements that would have to undergo change, and their performance is measured as the percentage of the total number of affected model elements that have been correctly identified. Fig. 2 shows an example of such a question for the VS system.

2.6.3 Postlab Surveys

In addition, because we wanted to gain enough insight to strengthen and explain our results, survey questionnaires were distributed after the execution of each *Defect Detection*, *Comprehension*, and *Maintainability* task. We asked each student whether he or she had enough time to perform the task, how easy the task was to perform, whether he or she understood the system's functionalities, or whether the lab instructions were clear and easy to follow. Moreover, for each student using OCL, questions were asked about the amount of time used to read OCL expressions, how easy they were to understand, and how helpful they were perceived to be. Most questions are answered on a *Likert scales* from 1 (Strongly agree) to 5 (Strongly disagree), following recommended templates [23]. An example questionnaire for the *Defect Detection* task with OCL is shown in Fig. 3.

The questionnaires are not part of the experiment proper, but rather gather information about the experiment. The output from the survey questionnaires is just meant to support and/or explain the quantitative results by providing qualitative insight.

2.7 Analysis Procedure

A two-step data analysis is performed on subjects' performance data. The first step accounts for all the observations of both attempts at the tasks (across two labs) for each dependent variable. However, it is also necessary

You are required to rate your level of agreement of the following statements. There are 5 levels of agreement:
1 – Strongly agree 2 – Agree 3 – Not certain 4 – Disagree 5 – Strongly disagree
Please mark the corresponding box dark to provide your level of agreement of each statement.

	1	2	3	4	5
1. I had enough time to read through the analysis document.	□	□	□	□	□
2. I had no problems to find errors in the document.	□	□	□	□	□
3. I would have found more errors if there would have been more time.	□	□	□	□	□
4. I fully understood what the system is trying to do.	□	□	□	□	□
5. I was not sure what type of errors I was looking for.	□	□	□	□	□
6. The system analysis was fairly complex.	□	□	□	□	□
7. The lab instruction is clear and easy to follow.	□	□	□	□	□
8. The OCL expressions in the document were easy to understand.	□	□	□	□	□
9. The OCL expression in the document helped me to find errors.	□	□	□	□	□

10.How much time did you spend in understanding the OCL expressions during this lab?
A. <25% B. >= 25% and <50% C. >=50% and < 75% D. >= 75%

Fig. 3. A survey questionnaire example.

to consider possible ordering effects, which cannot be avoided, and to determine whether they interfere with the overall analysis (e.g., learning effect). In other words, we want to test whether the difference between UML+OCL and UML-Only measurements can be explained by whether OCL is used during the first or the second attempt at a task. If this is the case, the data regarding each laboratory may have to be analyzed independently as we can expect different results. Thus, a second analysis step is performed for each dependent variable for each laboratory. In general, our statistical analysis will assume a level of significance $\alpha = 0.05$. Assumptions that have to be satisfied by the various analysis techniques we employ will be discussed in the next sections.

2.7.1 Step 1: Analyzing Data of Both Attempts at a Task

All observations analysis. As each subject performs each task twice, using and not using OCL, on each system, the *one-tailed paired* t-test for independent samples [11] is performed for each task, using all available observations for that task. If the derived p-value $\alpha = 0.05$, it can be concluded that using OCL has a significant positive effect on the corresponding *dependent variable*. Note that we have systematically performed a nonparametric statistical test as well for every hypothesis to be tested. The *Wilcoxon* test was used as an alternative to the t-test, as the *Wilcoxon* test is resilient to strong departures from the t-test assumptions. However, *Wilcoxon* test results are not reported (they can be found in [5]), since they are consistent with those of the t-test.

Testing effect of Order of Method. Half of the subjects apply the no_OCL treatment first and then the OCL treatment (referred to as order N_O), while the other half applies these treatments in the reverse order (referred to as order O_N). If there is a significant effect of *Order of Method*, it will then be necessary to study separately the effects of *Method* for each attempt at the tasks. To study the effect of *Order of Method*, the measurement differences between observations of the subjects when using and not using OCL are calculated, as follows:

$$Diff_x = observation_x(OCL) - observation_x(no\_OCL),$$

where x denotes a particular subject. $Diff(N\_O)$ denotes the performance difference ($Diff$) of the subjects who work on the system documents in the order N_O, while $Diff(O\_N)$ denotes the performance difference of the subjects who work on the system documents in the order O_N. We also use the symbol μ to denote averages of task performance or differences. A *one-tailed* t-test for independent samples [11] is performed to test whether $Diff(N\_O)$ is larger than $Diff(O\_N)$. $Diff(N\_O)$ is expected to be larger than $Diff(O\_N)$, as the subjects' ability to use OCL may improve through lectures and training received between laboratories. Therefore, $H_0 : Diff(N\_O) = Diff(O\_N)$, $H_a : Diff(N\_O) > Diff(O\_N)$. A p-value below $\alpha = 0.05$ would indicate that using OCL during the second attempt brings significantly larger performance differences when using and not using OCL than during the first attempt.

Performance comparison between the first and the second attempt at a task across Ability levels. In order to further understand the effect of *Order of Method* and learning effects, the *one-tailed paired* t-test is also performed to compare low and high-ability subjects' performance between the first and the second attempt for each specific task. If the subjects' performance improves over time, we want to determine whether there is a difference between low-ability and high-ability subjects. A p-value below $\alpha = 0.05$ for a particular *Ability* level would indicate that each subject performs significantly better during the second attempt than during the first.

2.7.2 Step 2: Data Analysis per Laboratory

When significant ordering effects take place, it is necessary to look into the data one laboratory at a time. To study the effect of *Method* in each laboratory, a *one-tailed* t-test for independent samples [11] for the *Method* factor would suffice. However, we expect student ability to have a strong effect on the task results. Further, *Method* may have a different effect (interaction) at different levels of subjects' ability. Though we control for *Ability* in our design, in order to refine our data analysis, we can thus use a *two-way* Analysis of Variance (ANOVA) [11], [16] using *Ability* as a factor as well. Likewise, though we try to minimize its effect by using comparable systems, a *three-way* ANOVA that includes the *System* variable could be considered as well. Unfortunately, in the first experiment trial, the available sample size does not permit *three-way* ANOVA. Instead, two-way ANOVA is performed for *Method and Ability*, and *Method and System*, respectively. A three-way ANOVA is performed for *Method*, *Ability*, and *System* in the second experiment trial, as the sample size is this time significantly larger.

The reason why two/three way ANOVA is better is two-fold [11]. First, ANOVA allows us to investigate interaction effects between *Ability* and the use of OCL (*Method*). Perhaps OCL has a different effect on students with different ability. Second, by introducing *Ability*, the variation it explains in the data is removed and the analysis of the effect of OCL becomes as a result statistically more powerful, e.g., we are more likely to see any significant effect if there is one.

When looking at ANOVA result tables, we want to determine whether sample means in the groups formed by *Ability* and *Method* categories (levels) are significantly different, i.e., that the independent variables have a significant impact on the dependent variables (*Defect Detection*, *Comprehension*, and *Maintainability*) and that the variations in means could not have been expected by chance alone. This is typically captured by the results of an F-test [11] that is yielding a p-value which captures the probability that we could obtain the results we observe if the populations from which the group samples are drawn had equal means (e.g., in terms of *Defect Detection*). Another standard statistic measures the strength of the relationship between each of the independent variables and the dependent variable. It is denoted as E^2, is referred to as the correlation ratio, and computes the relative improvement in our ability to predict the value of the dependent variable when we know what group an observation belongs to [16]—it is in many ways similar to R^2 in regression analysis. Correlation ratios are reported in Section 3.2 for

significant results. Note that, because our goal is not prediction, E^2 values need not be high for the results to be interesting. Their main purpose is to compare independent variables and their interaction in terms of explaining variation in the data.

The ANOVA procedure is based on a number of assumptions. Each combination of factor levels (e.g., OCL and high-ability) has to be assigned subjects in a random fashion (possibly using blocking as a constraint, as described in Section 2.5.3) and should also have an equal 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 [16], especially when the samples sizes are equal or similar for each level combination. If the differences are not too large, unequal sample sizes do not prevent the use of ANOVA, it just makes it more complex to interpret its results: The results may depend on the order in which factors are included in the model. Without further expanding on this, in our experiments the differences in sample sizes exist but are rather small [5]. As expected, departure from normality and equal variance assumptions exist, but a detailed analysis shows that differences in variance are not statistically significant [5] and the dependent variable distributions do not show extreme outliers or multimodal distributions, thus exhibiting only mild departures from normality. Moreover, we are in a situation where we are interested only in the effect of one variable (OCL) and the other factors are merely used to refine the analysis. In that case it is recommended to include the important variable last in the model, i.e., the factors are included in the model in order of increasing importance. In our case, *Method* is therefore the last variable to be included. In other words, we test to which extent OCL explains the remaining variation after the effects of *Ability* and *System* have been accounted for. Though the differences in sample sizes are very small in our case, we will follow this procedure to be on the safe side.

In order to help further the interpretation of results, *Descriptive statistics* tables and *Graph of means* diagrams are also provided either in the main text or [5] in addition to the ANOVA tables. They help assess and visualize the effects of factors and their interactions.

2.7.3 Analyzing the Survey Data

Survey questionnaires are used to detect differences between groups using UML+OCL and UML only (e.g., in terms of their perception of the usefulness of OCL). Such data are used to better explain and support our quantitative results. A simple t-test [11] for independent samples is used to compare groups. This test is one-tailed as we expect OCL to have an effect in a specific direction (e.g., questions are easier to answer and, therefore, exhibit better scores with OCL). When comparing differences in the central tendencies of questionnaire data between subsequent labs, our goal is to monitor for learning effects and explain differences in results that may be observed. Since groups of students are identical when comparing subsequent labs, we use a statistical test for paired samples, the *paired* t-test [11]. This test is also one-tailed as we expect a learning effect over subsequent laboratories and, therefore, a performance improvement in terms of questionnaire scores.

2.8 Replicated Experiment Trial (Experiment II)

A second experiment trial, referred to as Experiment II, is conducted, repeating the steps of the first experiment trial as an attempt to confirm its results (Table 2). The number of subjects is twice that of Experiment I (84 instead of 38), which enables a *three-way* ANOVA for the *Method*, *Ability*, and *System* factors. Due to classroom space constraints, each group has to be further divided into two subgroups, with each subgroup attending laboratories every two weeks [5]. Experiment II covers a time period of eight weeks (instead of four weeks for Experiment I).

More training was administered before the second experiment trial than in the first trial. This was motivated by our observation during the first trial that subjects needed to be better prepared, despite the courses they had already taken and passed so as to minimize the learning effects we had observed during the first experiment. More examples were given to the subjects to familiarize themselves with UML system analysis documents, OCL expressions, and the tasks at hand. It was also observed in Experiment I that, overall, the subjects did not use all the time allocated to perform the *Comprehension* task (90 minutes), while the subjects needed more time to perform the *Maintenance* task (60 minutes). Thus, 75 minutes are allocated to each of the two tasks in Experiment II.

Overall, the grade distributions are similar across the two experiment trials and the same blocking strategy was used for Experiment II [5]. However, the differences between Experiment I and Experiment II introduce new threats to *internal validity* that are discussed in Section 4.1.

3 Discussion of Results

This section discusses the analysis of the results for both experiment trials, according to the *two-step* analysis (Section 2.7). We will focus our discussion on factor *Method* as this is the main factor of interest in this article. More detailed results can be found in [5]. Section 3.1 reports on the first step of the analysis where we account for all the observations of both attempts at the tasks for each dependent variable. Section 3.2 focuses on the second step where we analyze each dependent variable for each laboratory. Section 3.3 summarizes the main observations.

In order to simplify the presentation of the tasks performed during the different laboratories in the different experiment trials, the following convention is used: <Task>-<Lab>-<Experiment Trial>. For example, Comprehension-2-I (or simply C-2-I) refers to the Comprehension task performed during Lab 2 in Experiment I. Also, in order to simplify the characterization of the subjects, a number of acronyms is used, as shown in Table 3.

We also use the symbol μ to denote averages of task performance or differences. Last, any p-value below the significant level $\alpha = 0.05$ in printed in bold face when reporting results.

TABLE 3
Types of Subjects

Types of subjects	Description
OCL subjects	The subjects who performed a task on the UML+OCL version of documents
No_OCL subjects	The subjects who performed a task on the UML-only version of documents
N_O subjects	The subjects who performed a task on the UML-only version of a document during the first attempt at the task, and on UML+OCL version of the document during the second attempt
O_N subjects	The subjects who performed a task on the UML+OCL version of a document during the first attempt at the task, and on UML-only version of the document during the second attempt

3.1 Analyzing Data of Both Attempts at Tasks

Table 4 summarizes the effects of using OCL. The rows represent the tasks performed in each experiment trial. There are two attempts for each task. The results of *both attempts* analysis and *each attempt* (per Lab) analysis are provided. The columns $\mu(OCL)$ and $\mu(no\_OCL)$ present the mean values of the subjects' performance using and not using OCL, respectively. The *p-value* columns report on the statistical significance of differences. In Experiment II, second attempt, when subjects were given more training and had the most experience, we see that the effect of OCL ($\mu(\mathrm{OCL}) - \mu(\mathrm{no\_OCL})$ in Table 4) is roughly of 5, 7, and 6 percent, for *Defect Detection*, *Comprehension*, and *Maintenance*, respectively.

We can also observe that the percentages (scores) of correctly detected defects and correctly answered questions are rather low (*Defect Detection*: < 12 percent; *Comprehension*: < 50 percent; *Maintenance*: < 50 percent), and that *Comprehension* appears to be simpler than *Maintenance* and *Maintenance* appears to be simpler than *Defect Detection*. The low performance may be due to a lack of experience with the tasks or with their inherent complexity. Even after having gone through thorough UML analysis and design training, fourth year engineering students may not be comfortable with handling relatively large system analysis models and documents of such complexity. This clearly tells us these are difficult tasks that should perhaps be performed by teams, not individuals. In particular, these results suggest that there are perhaps significant gains in better supporting the inspection of models for defect detection and impact analysis in UML models [6].

TABLE 4
Effect of Using OCL

		Both attempts			Each attempt		
	Task	μ(OCL)	μ(no_OCL)	p-value	μ(OCL)	μ(no_OCL)	p-value
Experiment I	Defect Detection (1st attempt: D-1-I)	11.67%	10.28%	0.30	7.3%	7.1%	0.47
	Defect Detection (2nd attempt: D-3-I)				14.9%	14.23%	0.44
	Comprehension (1st attempt: C-2-I)	49.29%	42.86%	**0.03**	41.7%	45.9%	0.85
	Comprehension (2nd attempt: C-4-I)				57.9%	39.65%	**0.002**
	Maintenance (1st attempt: M-2-I)	36.78%	31.76%	0.16	28%	32%	0.8
	Maintenance (2nd attempt: M-4-I)				46%	31%	**0.02**
Experiment II	Defect Detection (1st attempt: D-1-II)	9.48%	7.76%	**0.05**	7.5%	8.13%	0.66
	Defect Detection (2nd attempt: D-3-II)				11.9%	7%	**0.003**
	Comprehension (1st attempt: C-2-II)	49.87%	42.36%	**0.0001**	46%	41%	**0.02**
	Comprehension (2nd attempt: C-4-II)				51%	44%	**0.01**
	Maintenance (1st attempt: M-2-II)	41.37%	33.25%	**0.003**	40%	31%	**0.003**
	Maintenance (2nd attempt: M-4-II)				41%	35%	**0.04**

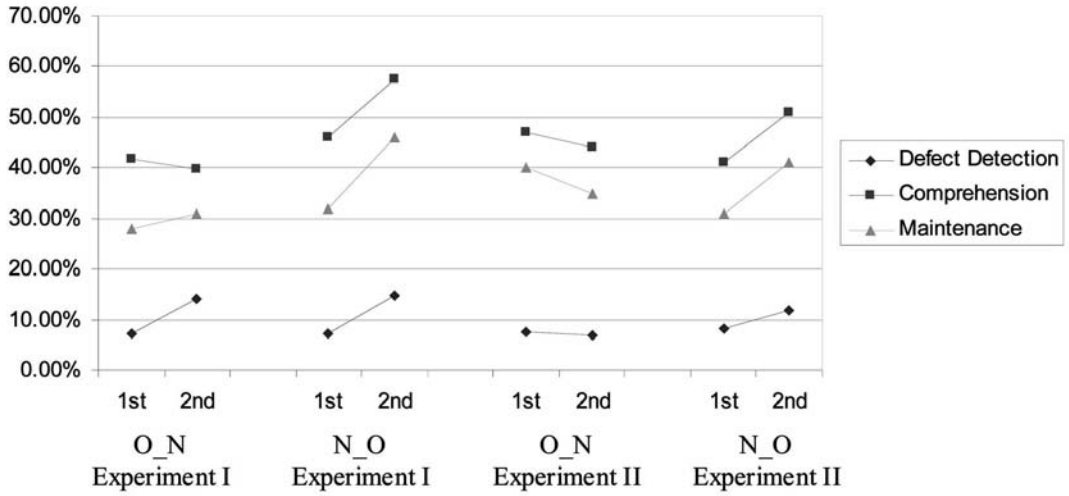

Fig. 4. Visualizing learning effects (y-axis shows the percent of correct answers).

In Experiment I, when considering data from both attempts at the tasks together, using OCL has a significant positive effect only for the *Comprehension* task, but not for the *Defect Detection* or the *Maintenance* tasks. This difference may be explained by the fact that the *Comprehension* task is easier and requires less skills (as visible from performance scores). When considering data from each attempt, however, using OCL has positive effects for both *Comprehension* and *Maintenance* tasks only for the second attempt. This is interpreted as resulting from a learning effect taking place between laboratories, as people are more familiar with the tasks, UML and OCL during their second attempt at a task. Although using OCL does not have any significant positive effect on the *Defect Detection* task, it has significant interaction effects with ability in the second attempt as further discussed below.

On the other hand, in Experiment II, using OCL has significant positive effects for each task, whether taking all observations into account, or considering data from each attempt, except for the first attempt at the *Defect Detection* task (the first task of the experiment). Further analysis shows that, in the second attempt at the Defect Detection task, using OCL also has significant interaction effects with ability as further discussed below. Therefore, given a context where software engineers are adequately trained, there is strong evidence to suggest that using OCL results in statistically significant benefits in terms of *Defect Detection*, *Comprehension*, and *Maintenance*. In terms of practical significance, the differences between percentages of correct answers are not substantial (5-7 percent) but it is important to realize that those benefits accumulate and that there are probably many other tasks in which OCL can help besides the ones we experimented with. Furthermore, with even more training, experience, and better tool support, the positive effect of OCL could keep increasing. The case of defect detection is the most complex, as benefits depend on system characteristics, subjects' ability, and possibly other factors. Recall that the subjects of Experiment II received a more thorough prior training in OCL and UML than those of Experiment I. This may have resulted in weaker learning effects in Experiment II and probably explain why *Method* had significant main effects only for the last two tasks of Experiment I whereas these effects are significant for most of the tasks in Experiment II. The results indicate that using OCL can bring significant benefits but that the required training in using OCL and UML may be substantial and, therefore, expensive.

In order to further look into learning effects, Fig. 4 illustrates differences from the first attempt at a task to the second, for each experiment trial. In Experiment I, we observe strong learning effects, as performance improvements are observed for all but one task, regardless of the order in which OCL is used. In Experiment II, performance improvements are observed only when OCL is used in the second attempt (N_O), but not in the first attempt (O_N). In Experiment II milder learning effects tend to cancel out OCL effects in the O_N cases whereas the two effects add up in the N_O case. In Experiment I, the learning effects are stronger and override the OCL effects to such an extent that some slight improvements are even observed in the O_N cases for *Defect Detection* and *Maintenance*. There is also evidence of learning effect in the survey data of each experiment trial, indicating for examples that the subjects are significantly more confident in the types of defects to look for in Lab 3 than in Lab 1 (p-value = 0.03 in each trial).

Note that the only significant performance improvement is experienced by those subjects who use OCL in the (N_O) order as shown in Table 5. This table also shows that using OCL seems to benefit high-ability subjects in some instances and low-ability subjects in other instances. This indicates complex interactions between OCL usage and modeling ability, which can be investigated only via ANOVA as discussed below.

3.2 Data Analysis per Laboratory

Table 6 summarizes the ANOVA results for the three factors: *Method*, *Ability*, and *System*, for each laboratory of each experiment trial. Rows present main and interaction effects and columns present tasks. For each task, if an effect is significant, the *correlation ratio* (E^2) is reported in the corresponding cell.

TABLE 5
Significant Improvement by the Subjects

	Task	Subjects with significant performance improvement from 1st attempt to 2nd attempt
Experiment I	Defect Detection	Low-ability, N_O
	Comprehension	High-ability, N_O
	Maintenance	(None)
Experiment II	Defect Detection	High-ability, N_O
	Comprehension	High and low-ability, N_O
	Maintenance	Low-ability, N_O

First, ANOVA confirms that *Method* has significant main effects in C-4-I, M-4-I, D-3-II, C-2-II, C-4-II, M-2-II, and M-4-II. It also shows that *Ability* plays an important role: *Ability* has significant main effect in D-1-I, D-3-I, C-4-I, M-4-I, D-1-II, D-3-II, and C-4-II. In those laboratories, high-ability subjects perform better than low-ability subjects, irrespective of the methods used. An individual's ability thus appears to be key to such software engineering tasks. The survey results explain the existence of a significant effect of *Ability*. In D-1-II for instance, compared to low-ability subjects, high-ability subjects have less problems to find defects and they know better what to look for in finding defects. *System* also appears to have a significant main effect (in D-3-I, C-4-I, M-2-I, M-4-I, C-1-II, C-4-II, M-2-II, M-4-II) and more particularly so for *Maintenance* tasks. Subjects consistently perform better for the CD system than for the VS system in those laboratories.

Method has significant interaction effects with *Ability* in D-3-I, D-3-II, and M-2-II. There are inconsistencies in the interactions between *Method* and *Ability* for *Defect Detection* (Fig. 5): Using OCL hinders high-ability subjects in D-3-I, but helps them in D-3-II, while OCL helps low-ability subjects in D-3-I but does not have any significant effect in D-3-II $\mu(\mathrm{OCL}) = 8.6$, $\mu(\mathrm{no\_OCL}) = 6.83$). However, based on survey results, high-ability subjects perceived OCL to be useful more than low-ability subjects in both D-3-I and D-3-II (for example in D-3-II: $\mu(\mathrm{High}) = 2.56$, $\mu(Low) = 3.05$, $\text{p-value} = 0.047$). A possible interpretation is that when they are not sufficiently familiar with OCL, high-ability subjects do not benefit from OCL to find defects, as their insight and intuition with respect to UML system analysis is sufficient. Reading and analyzing OCL expressions therefore results in a time overhead without real advantage. On the other hand, low-ability subjects tend to rely on OCL expressions and use them in a systematic way to detect inconsistencies in UML system analysis documents. Once more, experience is gained in using OCL and UML system analysis, high-ability subjects eventually benefit from using OCL and perhaps to a greater extent than low-ability students as they learn faster. In any case, it is clear that the interactions among learning effects, ability, and the use of OCL must be further investigated as we see that complex phenomena are taking place.

In M-2-II, using OCL seems to have a much greater positive effect on low-ability subjects than on high-ability subjects ($Diff(\mathrm{High}) = 7$ percent, $Diff(\mathrm{Low}) = 13$ percent): Fig. 6. Similar to D-3-I, this may be due to high-ability subjects being able to rely more on their intuitive understanding of the UML model, then benefiting less from using

TABLE 6
E^2 for Significant Effects ($\alpha = 0.05$)

Experiment I						
	D-1-I	**D-3-I**	**C-2-I**	**C-4-I**	**M-2-I**	**M-4-I**
Method				0.27		0.13
Ability	0.15	0.20		0.12		0.22
System		0.31		0.18	0.26	0.47
Method & Ability		0.19				
Method & System						

Experiment II						
	D-1-II	**D-3-II**	**C-2-II**	**C-4-II**	**M-2-II**	**M-4-II**
Method		0.11	0.07	0.07	0.09	0.05
Ability	0.06	005		0.06		
System	0.11			0.07	0.39	0.49
Method & Ability		0.04			0.05	
Method & System		0.08			0.05	
Method, Ability & System						

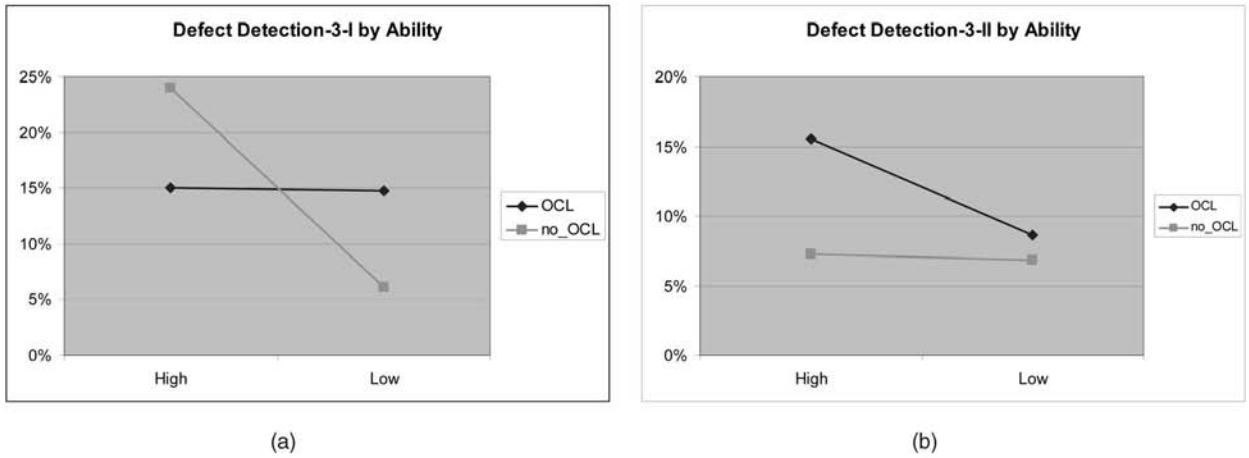

Fig. 5. Defect Detection (y-axis) by Ability (x-axis) in each Experiment Trial (Lab 3).

OCL than low-ability subjects for the *Maintenance* task. However, this interaction between *Method* and *Ability* is not visible in M-4-II. It is possible that learning effects with respect to UML system analysis and the *Maintenance* task help low-ability subjects catch up with high-ability subjects, thus resulting in similar OCL effects for high-ability subjects and for low-ability subjects.

Method also has significant interaction effects with *System* in D-3-II and M-2-II: Fig. 7. In D-3-II, using OCL results in a larger performance improvements for the VS system than for the CD system (Diff(VS) = 10 percent, Diff(CD) = 0.6 percent). This suggests that using OCL may have a larger impact on *Defect Detection* for more complex system models (such as VS) as this is in the context of such complexity that formality pays off. This is visible only in the context of Experiment II as such benefits may be obtained only if subjects have been provided with enough training. On the other hand, in M-2-II, using OCL results in a larger performance improvement for the CD system than for the VS system (Diff(CD) = 17 percent, Diff(VS) = 3.9 percent). This suggests that using OCL may have larger impact on *Maintenance* for simpler system models. However, this interaction is not visible in M-4-II. The interaction effect between *Method* and *System* on different software engineering activities, such as detecting defects and maintenance, needs to be investigated further in future experiments.

3.3 Discussion

The results of this study show that OCL, as a constraint language complementary to UML, is potentially useful in helping understand system models, facilitate change, and detect defects in such models. Though the effects are modest (5 to 7 percent increase, as described in Table 4) in each task, these benefits are cumulative and it is expected that 1) other tasks would benefit from OCL and 2) benefits will grow overtime as people acquire more UML and OCL expertise. However, the interaction effects of subjects' modeling ability and system complexity with the usage of OCL need to be further studied and understood. Furthermore, the benefits of using OCL may be observable only if subjects have substantial training and experience in UML and OCL. We believe that such phenomena are likely to be encountered when using any formal method and given their complexity, it is then perhaps not surprising that the benefits of using formal methods has been so controversial [3], [25], [27].

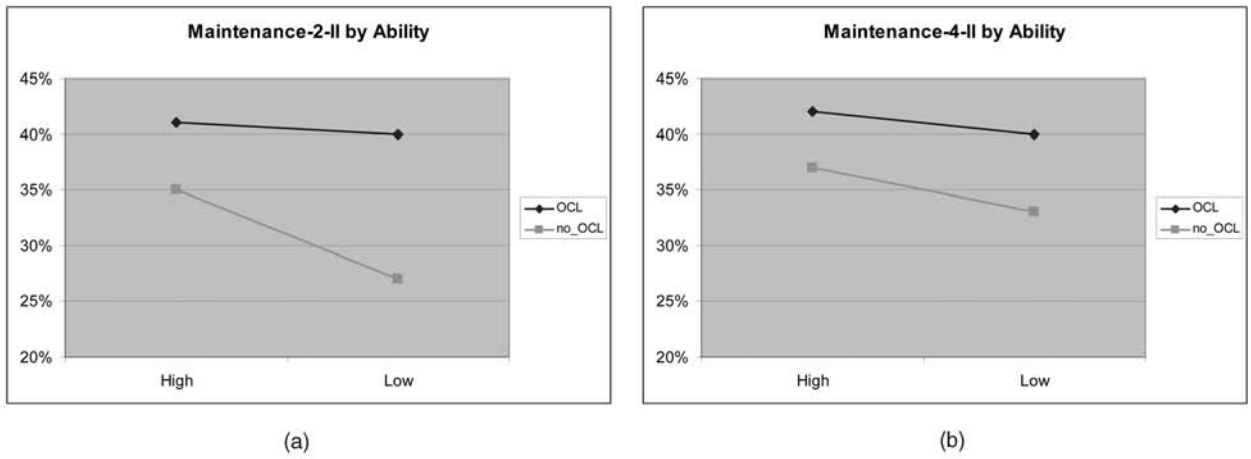

Fig. 6. Maintenance (y axis) by Ability (x axis) in Experiment II.

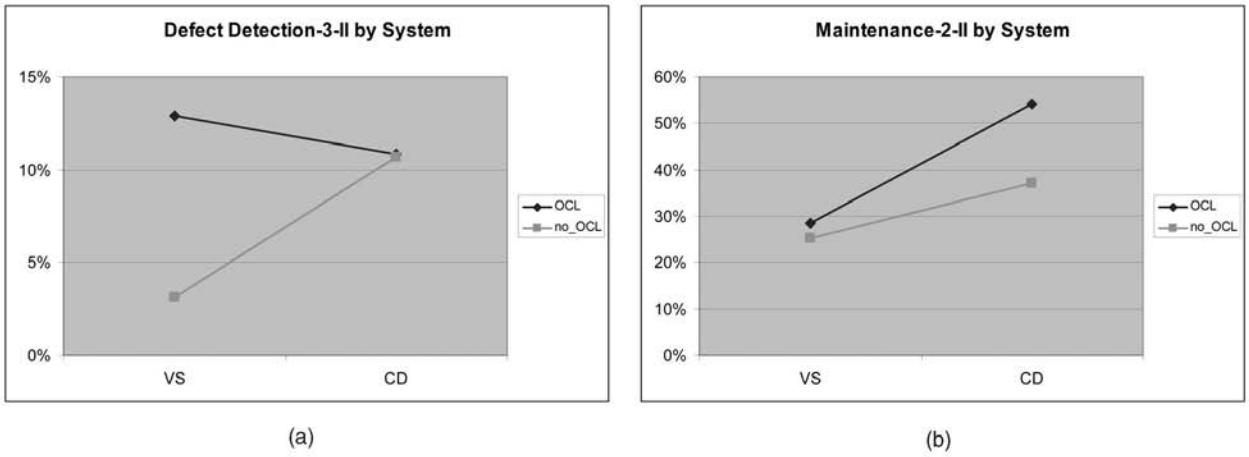

Fig. 7. Interaction between Method and System (x axis) in Experiment II (y axis shows the percent of correct answers).

4 Threats to Validity

This section further discusses the different types of threats to the validity of our experiments, in order of decreasing priority: *internal*, *external*, *construct*, and *conclusion validity* [9], [29].

4.1 Internal Validity

One possible issue related to internal validity, is due to the experiment design and concerns the possible information exchange among the subjects between the laboratories. This is prevented in several ways, as discussed in Section 2.5.3 (e.g., subjects were not allowed to communicate with each other during laboratories, and were required to hand in all experimental material at the end of each laboratory). Another concern relates to the fact that the subjects perform two distinct tasks on the same system document in two subsequent laboratories. If the subjects remember the OCL expressions from the first laboratory, it may help them to perform better in the following laboratory. After considering other alternatives, we concluded the final design selected was the least problematic as it was unlikely that the subjects would remember the OCL expressions from the first laboratory: the laboratories are one week apart in Experiment I, and two weeks apart in Experiment II. Finally, the subjects do not know what task to expect in the next laboratory and have no access to lab material between the laboratories.

Another issue relating to internal validity is that the absence or presence of the subjects cannot be fully controlled: Students may drop out of the course, or simply not show up for laboratories. Fortunately, this happened only rarely and did not significantly affect the grade distributions in the groups.

Finally, new threats to internal validity are introduced with Experiment II. Since the subjects have to be divided into two subgroups and perform the same tasks in two subsequent laboratories, it is possible that the subjects from the second subgroup may hear from the first subgroup about the tasks and the analyzed systems. Problems were actually observed in a few cases, as the students were monitored, and the corresponding data were discarded. Further, when comparing the performance of the subgroup pairs, no significant differences were observed.

4.2 External Validity

The main threats to validity in this experiment relate to external validity. Like any academic experiment, the issue of whether the subjects are representative of software professionals arises. Recall that—and it is unusual for student experiments—the fourth year engineering students involved in this experiment are probably better trained at software modeling with the UML and OCL than most software professionals. So, if there is an issue, it is not the lack of knowledge and experience of our subjects, but rather that they represent the ideal, unusual case of subjects having been through thorough UML training, but without having practiced their knowledge in an industrial environment. However, an increasing number of software engineering graduates with such modeling skills are now integrating the software industry and should therefore become, slowly but surely, the norm. Moreover, the difference between students and professionals is not always clear cut, as reported in a recent experiment [1]: The performance of undergraduate and graduate students (whose expertise is similar to the one of our students) at maintaining a UML design was not found to be very different from the performance of junior and intermediate professional consultants.

Another external validity issue, which is unfortunately inherent to controlled experiments, is the size and complexity of the system models. The question is then whether OCL would have an equal, weaker, or stronger impact had the system models been vastly more complex. We cannot answer that question and we see controlled experiment such as the one presented here as a first step before assessing whether actual case studies in industry settings should be even considered. Our intuition though is that OCL would be even more needed on increasingly complex systems as there would be more room for ambiguity, larger teams would be involved in developing the models, and it would be increasingly more difficult for software engineers to rely on intuition.

4.3 Construct Validity

Construct validity relates to the measurement instruments, including the defects seeded, the *Comprehension* and

Maintenance questionnaires, and the survey questionnaires. The seeded defects were selected to achieve a balance: They have to be complex enough so as not to be obvious at a first sight; they also have to be detectable with or without OCL so as not to bias the results in favor of using OCL. The survey questionnaire [5] indicated that this point was addressed, i.e., defects did not appear to be trivial, and were not impossible to be found. The *Maintenance* and *Comprehension* questions were also selected to be of certain level of complexity [5] and to cover as many parts of the models as possible, while being answerable within the time available. Questions also had to be answerable with or without OCL so as not to bias the results in favor of using OCL. Questions were phrased so as to be answered either as multiple choices questions (*Comprehension*) or by providing a list of model elements (*Maintenance*), thus both leading to an easy measurement of the subjects' performance. The fact that tasks can be performed with or without OCL is however a conservative decision as questions that require OCL to be answered may actually occur in practice. Seeded defects and questionnaires were produced by one of the authors, who was not involved in the modeling of the two systems.

The survey questionnaires focus on capturing the perception of the subjects on their tasks. They are not meant to statistically show the impact of the independent variable (*Method*) on our dependent variables but, rather, our objective is to support and explain our quantitative results by providing qualitative insight from the questionnaire data. We followed standard ways of designing survey questionnaires [23] and phrase subjective questions so as to avoid bias and increase reliability of answers. However, a possible bias could have been the following: When asked about their perception of the usefulness of OCL, students could have answered what they thought was expected by the experimenters. Evidence shows this was, however, not the case [5].

4.4 Conclusion Validity

Conclusion validity relates to subject selection, data collection, measurement reliability, and the validity of the statistical tests. These issues have been addressed in the design of the experiment through a careful design of questionnaires and fault selection, and appropriate analysis techniques (Section 2.5).

5 Conclusions and Work-in-Progress

This paper investigates an important methodological aspect of the use of the Unified Modeling Language (UML) [4] for Object-Oriented Analysis and Design: whether or not the Object Constraint Language (OCL) [28], which is part of the UML standard, should be used to augment the precision and rigor of system analysis modeling. This implies, at a minimum, to use OCL to specify class invariants, operation preconditions and postconditions. Conditions in other diagrams can also be defined using OCL, as in the current study. In the specific context of UML, this question is very much akin to the ongoing debate about the use of formal methods in software development [2], [25]. It is an important, practical question as there is a debate about the level of formality by which UML should be used [12], [18], [28].

In order to investigate the impact of OCL in UML development, we designed, performed, and replicated a controlled experiment. It involved fourth year software/computer engineering students who received substantial training in UML and OCL. We investigated the impact of using OCL on three important software engineering activities: 1) understanding the functionality and internal logic of modeled systems, 2) performing a change impact analysis based on UML models, and 3) detecting defects through model inspections. It was important to ensure that the controlled experiment would yield valid results by making sure the trends we observe are due to the use of OCL and not other extraneous factors. To this end, a careful experiment design was devised to prevent confounding effects, such as a widely varying modeling ability among students and differing system model properties. In order to yield conservative results, each task is designed such that questions can be answered, and defects can be detected, with or without OCL. Further, not only the effect of using OCL was examined, but also its interactions with others factors were carefully analyzed.

The results of the experiments show that using OCL has the potential to significantly improve an engineers' ability to understand, inspect, and modify a system modeled with UML. However, the benefits for each task are modest and become practically significant only when taken all together. Furthermore, such benefits are strongly dependent on the ability, experience, and training of software engineers. This is not entirely surprising as it has been often observed that new methodologies or technologies take substantial time to assimilate before they start paying off [24]. From a practical perspective, this means that it is unlikely that software engineers will benefit from using OCL in UML models unless they are properly trained and mentored. Based on our experience, this requires that the level of training be much more substantial than what is typically observed across the software industry. But, this is also likely to become increasingly easier as new software engineering graduates, who have received substantial UML and OCL training, permeate software development organizations and as the UML standard attains a firmer penetration. In light of those conclusions, it is then not surprising that popular textbooks [12], [18] recommend against using OCL as it is probably the best thing to do in most environments where the expertise is inadequate and no sufficient training is available. To come back to the original, more general issue of using formal methods, if what we have observed here can be generalized to other notations and techniques, it is also not then surprising that existing results appear inconclusive [14], [25]: So many factors are affecting the outcome of formal methods that we have to expect widely varying benefits depending on contextual factors such as training, experience, and tools. Because, like many other formal specification methods, OCL is based on first-order logic and set theory, we further believe that the results of our experiments have no reason to be specific to OCL and would very likely be observed regardless of the specific language used.

Although the presence of OCL in UML models has the positive effects reported in this paper, future experiments should determine whether the benefits of devising OCL expressions justify their cost. There are, however, other considerations which need to be taken into account, such as how to measure cost, the quality of implementation, and the correctness and completeness of UML models. Furthermore, the cost and benefit of using OCL constraints also depends on tool support to verify the correctness, completeness, and consistency of OCL expressions. As such tools emerge, the cost-benefit of defining precise UML models with OCL constraints should be re-evaluated.

When the experiments reported above were performed, UML 2.0 [22] and OCL 2.0 [21] were not released yet as standards and were in a rough draft state when we started designing the material. We do not expect, however, that the changes in the standards would have a dramatic impact on the results for the case studies that were used, especially regarding OCL that has retained most of its earlier features. Further experiments may nevertheless be warranted that would use these newly released standards.

ACKNOWLEDGMENTS

This work was partly supported by a Canada Research Chair (CRC) grant. Lionel Briand and Yvan Labiche were further supported by NSERC operational grants. This work is part of a larger project (http://www.sce.carleton.ca/Squall). The authors would like to thank the students who participated in the experiment. The authors would like to thank Daniel Berry for his comments on an early version of this article.

REFERENCES

[1] 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.

[2] D.M. Berry, "Formal Methods, the Very Idea, Some Thoughts on Why They Work When They Work," *Science of Computer Programming,* vol. 42, no. 1, pp. 11-27, 2002.

[3] D.M. Berry and W.F. Tichy, "Comments on "Formal Methods Application: An Empirical Tale of Software Development," *IEEE Trans. Software Eng.,* vol. 29, no. 6, pp. 567-571, June 2003.

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

[5] L.C. Briand, Y. Labiche, M. Di Penta, and H.-D. Yan, "An Experimental Investigation of Formality in UML-Based Development," Technical Report SCE-03-22, Carleton Univ., http://www.sce.carleton.ca/Squall, Jan. 2004.

[6] L.C. Briand, Y. Labiche, L. O'Sullivan, and M. Sowka, "Automated Impact Analysis of UML Models," *J. Systems and Software,* to be published.

[7] L.C. Briand, Y. Labiche, and H. Sun, "Investigating the Use of Analysis Contracts to Improve the Testability of Object-Oriented Code," *Software—Practice and Experience,* vol. 33, no. 7, pp. 637-672, 2003.

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

[9] D.T. Campbell and J.C. Stanley, *Experimental and Quasi-Experimental Designs for Research.* Houghton Mifflin, 1990.

[10] S. Cook, A. Kleppe, R. Mitchell, B. Rumpe, J. Warmer, and A. Wills, "Amsterdam Manifesto on OCL," *Object Modeling with the OCL, The Rationale Behind the Object Constraint Language,* 2002.

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

[12] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language,* third ed. Addison Wesley, 2003.

[13] H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML.* Addison Wesley, 2000.

[14] A. Hall, "Using Formal Methods to Develop an ATC Information System," *IEEE Software,* vol. 13, no. 2, pp. 66-76, 1996.

[15] P. Henderson, "Mathematical Reasoning in Software Engineering Education," *Comm. ACM,* vol. 46, no. 9, pp. 45-50, 2003.

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

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

[18] C. Larman, *Applying UML and Patterns—An Introduction to Object-Oriented Analysis and Design and the Unified Process.* second ed. Prentice Hall, 2002.

[19] B.H. Liskov and J.M. Wing, "A Behavioral Notion of Subtyping," *ACM Trans. Programming Languages and Systems,* vol. 16, no. 6, pp. 1811-1841, 1994.

[20] OMG, "Unified Modeling Language (UML),"Object Management Group V1.4, www.omg.org/technology/uml/, 2001.

[21] OMG, "OCL 2.0 Specification,"Object Management Group, Final Adopted Specification ptc/03-10-14, 2003.

[22] OMG, "UML 2.0 Superstructure Specification,"Object Management Group, Final Adopted Specification ptc/03-08-02, 2003.

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

[24] S.L. Pfleeger, "Understanding and Improving Technology Transfer in Software Engineering," *J. Systems and Software,* vol. 47, nos. 2-3, pp. 111-124, 1999.

[25] S.L. Pfleeger and L. Hatton, "Investigating the Influence of Formal Methods," *Computer,* vol. 30, no. 2, pp. 33-43, 1997.

[26] M. Satpathy, C. Snook, R. Harrison, M. Butler, and P. Krause, "A Comparative Study of Formal and Informal Specifications of a Case Study from Industry," *Proc. IEEE/IFIP Joint Workshop Formal Specifications of Computer-Based Systems,* pp. 133-137, Apr. 2001.

[27] A.E.K. Sobel and M.R. Clarkson, "Formal Methods Application: An Empirical Tale of Software Development," *IEEE Trans. Software Eng.,* vol. 28, no. 3, pp. 308-320, Mar. 2002.

[28] J. Warmer and A. Kleppe, *The Object Constraint Language,* Addison-Wesley, http://www.klasse.nl/english/boeken/errata.html, 1999.

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

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 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 coeditor-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 *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.

Yvan Labiche received the BEng degree in computer system 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 doing his PhD 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 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 Computer Society and has been on the program committee of IEEE conferences such as ISSRE, ICSM, and MoDELS.

Massimiliano Di Penta received the laurea degree in computer engineering in 1999 and the PhD degree in computer engineering in 2003 at the University of Sannio in Benevento, Italy. Currently, he is assistant professor at the same University and leading researcher at RCOST —Research Centre On Software Technology. His main research interests include software maintenance, reverse engineering, program comprehension and service-centric software engineering. He has authored more than 50 papers among journals, conferences, and workshops. He is member of IEEE and IEEE Computer Society, and has served program and organizing committees of conferences and workshops such as, CSMR, GECCO, ICSM, IWPC, SCAM, STEP, WCRE, and WSE.

Han (Daphne) Yan-Bondoc received the BEng degree in computer application from Shanghai University of Engineering Science (Shanghai, China). She became a software designer in a software company after the graduation, and then became an information techonology specialist for a Swiss bio company Novartis, responsible for the company's computer network in China. She joined Nortel as a quality engineer in 1999, and meanwhile obtained her master's degree in computer science in 2004 (Carleton University, Ottawa, Canada).