# On the Impact of UML Analysis Models on Source-Code Comprehensibility and Modifiability

GIUSEPPE SCANNIELLO, University of Basilicata
CARMINE GRAVINO, University of Salerno
MARCELA GENERO and JOSE' A. CRUZ-LEMUS, University of Castilla-La Mancha
GENOVEFFA TORTORA, University of Salerno



We carried out a family of experiments to investigate whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code. The family consists of a controlled experiment and 3 external replications carried out with students and professionals from Italy and Spain. 86 participants with different abilities and levels of experience with UML took part. The results of the experiments were integrated through the use of meta-analysis. The results of both the individual experiments and meta-analysis indicate that UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability.

Categories and Subject Descriptors: D.2.0 [**Software Engineering**]: General; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; D.3.2 [**Programming Languages**]: Language and Classification

General Terms: Documentation, Design, Experimentation, Human Factors

Additional Key Words and Phrases: Analysis models, UML, controlled experiment, family of experiments, maintenance, comprehensibility, modifiability, replicated experiments

**ACM Reference Format:**
Giuseppe Scanniello, Carmine Gravino, Marcela Genero, Jose' A. Cruz-Lemus, and Genoveffa Tortora. 2014. On the impact of UML analysis models on source-code comprehensibility and modifiability. ACM Trans. Softw. Eng. Methodol. 23, 2, Article 13 (March 2014), 26 pages.
DOI: http://dx.doi.org/10.1145/2491912

## 1. INTRODUCTION

The Unified Modeling Language (UML) [OMG 2010] is the de facto standard for object-oriented software analysis and design modeling [Erickson and Siau 2007; Grossman et al. 2005]. However, there is still significant resistance to model-based development in many software organizations, because UML is perceived to be difficult to learn and use for novice developers [Agarwal and Sinha 2003], expensive, and not necessarily cost-effective [Arisholm et al. 2006]. This is even worse for organizations that use lean processes to develop software [Cohen et al. 2004]. It is thus important, if not crucial, to investigate whether the use of UML can make a practical difference and justify the implied costs. It is also important to study in which context and under which conditions UML makes or does not make a practical difference.

This research has been partially funded by GEODAS-BC project (Ministerio de Economa y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01). Corresponding author's email: giuseppe.scanniello@unibas.it.

DOI: http://dx.doi.org/10.1145/2491912

Although there are a number of empirical studies on UML [Budgen et al. 2011], few evaluations of the benefits derived from its use in the entire software development life cycle have been reported [Anda et al. 2006]. This lack is even more evident in the software maintenance phase with respect to the benefits of UML models in source-code comprehension and modification. The results of the survey presented by Scanniello et al. [2010] showed that the core business of the interviewed companies mainly is concerned with the development and maintenance of software systems modeled using UML and implemented using object-oriented programming languages. With regard to the maintenance phase, very often the software maintainers have at their disposal only the models produced in the requirements analysis process. The survey presented by Dobing and Parsons [2006] showed that many of the respondents (e.g., members of the OMG organization) found the models produced in the requirements elicitation phase useful for performing maintenance operations. In particular, 68% and 61% of the respondents rated, respectively, use case narratives and use case diagrams as useful for software maintenance. Weidenhaupt et al. [1998] claimed that in industry, use cases are not only useful in the requirements engineering phase, but in the whole system development process, including software maintenance. The results were obtained from application scenarios in 15 industrial projects. In an early paper [Lubars et al. 1993], the results of a series of structured interviews with the practitioners are presented. This study aimed to find out how software organizations deal with the definition, interpretation, analysis, and use of requirements for their software systems and products. These practitioners were employees of 10 software organizations who worked on 23 projects from different domains. One of the most interesting results is that models produced in the requirements engineering process are used in software reuse and integration. Based on these findings, we have therefore performed a series of empirical investigations (i.e., a family of controlled experiments) with the goal of verifying the following research question.

Do the software models produced in the requirements analysis process aid in the comprehensibility and modifiability of source code?

To obtain an initial insight into the usefulness of these models, some of the authors of this article carried out an experiment as a pilot study (with 16 third-year bachelor students from the University of Basilicata in Italy) [Gravino et al. 2010]. They considered the method proposed by Bruegge and Dutoit [2003], in which functional requirements are represented by functional models, object (or conceptual) models, and dynamic models. Use case diagrams and use cases were employed to represent functional requirements. Class diagrams were used to abstract the objects from the problem domain (i.e., the object or conceptual model), while sequence diagrams were employed to model the dynamic and/or functional behavior of both the users and the system. For brevity, in the remainder of this article, we shall use the term *UML analysis models* (or analysis models) to indicate these models.

The results of the pilot study revealed that the comprehension of source code slightly improves when it is added with UML analysis models (about 1%). In order to increase external validity, we carried out a family of experiments to investigate whether this result also holds in different contexts. The family does not include the pilot study [Gravino et al. 2010] and consists of one experiment (conducted in Italy at the University of Basilicata with students) and three external replications (performed in Spain at the University of Castilla La Mancha with students and practitioners). In order to take into account the differences between experiments and to obtain the overall effect of the analysis models, we have integrated the results of the experiments by performing a meta-analysis.

Table I. Participants in the Family of Experiments Grouped by Experiment

| Experiment | Context | Description | Number | Type |
|---|---|---|---|---|
| E-UBAS | University of Basilicata | Original experiment | 24 | 1st year MSc Students |
| R1-UCLM | University of Castilla La Mancha | Replication of E-UBAS | 22 | 2nd year MSc Students |
| R2-UCLM | University of Castilla La Mancha | Replication of E-UBAS | 22 | 1st year MSc Students |
| R3-UCLM | Practitioners in Spain | Replication of E-UBAS | 18 | - |

This article is organized as follows. In Section 2, we present the family of the experiments, while the results obtained are presented in Section 3 and discussed in Section 4. Threats to validity and related works are highlighted in Section 5 and Section 6, respectively. The article concludes with our final remarks and future work.

## 2. THE FAMILY OF EXPERIMENTS

Families of experiments allow researchers to answer questions that are beyond the scope of individual experiments and to generalize findings across studies, thus providing evidence with which to confirm or reject specific hypotheses [Basili et al. 1999]. In addition, families of experiments can contribute to the conception of important and relevant hypotheses that may not be suggested by individual experiments.

To show that a given finding is robust, external replications[1] can be conducted to get additional confidence that the original results were not affected by experimenter bias. The choice and the number of factors to be varied is relevant, because a larger number of variations could make it less likely that observed results are traced to the factor of interest [Shull et al. 2008]. Variations in the experience of the participants and in the environmental factors in replications could contribute some confidence that the effect is not limited to one particular setting.

We carried out a family of experiments consisting of an experiment and three external replications. Table I summarizes the experiments of our family. The original experiment (denoted E-UBAS) was carried out at the University of Basilicata in 2010 with 24 first-year students from the Master's Program in Computer Science. This experiment was replicated three times at the University of Castilla La Mancha in Spain in the same year by varying the experience of the participants. These latter experiments were denominated as R1-UCLM, R2-UCLM, and R3-UCLM. The participants in R1-UCLM were 22 second-year students from the Master's Program in Computer Science. R2-UCLM was performed with 22 first-year students from the Master's Program in Computer Science. R3-UCLM was conducted with a group of 18 practitioners. All the professionals had at least a bachelor degree in Computer Science.

Features that made the experiments in the family distinct from the pilot study were the experimental design and the dependent variables used. We also renewed and improved the material and experimental objects. The experimental material used in the pilot and in E-UBAS was in Italian. The replications were performed after a native Spanish speaker had translated all the material (e.g., documentation and identifiers of the source code) from Italian into Spanish. All of the participants in our family of experiments had more experience than the people who took part in the pilot study presented by Gravino et al. [2010]. The data of the pilot study are not analyzed together with those of the family of experiments presented here.

The experiments were carried out by following the recommendations provided by Juristo and Moreno [2001], Kitchenham et al. [2002], and Wohlin et al. [2000]. The experiments were reported according to the guidelines suggested by Jedlitschka et al.

[1]These experiments can also be considered differentiated replications of the original experiment. In fact, they introduce a variation in the essential aspects of the experimental conditions, that is, the kind of participants involved.

[2008]. For replication purposes, we made an experimental package available on the Web at http://www2.unibas.it/gscanniello/ExpAMvsSC/. This package also includes the raw data of the experiments and additional analyses.

In the following sections, we describe the experimental process followed to carry out the family of experiments.

### 2.1. Goal

The goal of our family of experiments can be formalized as follows, according to the GQM (Goal Question Metrics) template by Basili and Rombach [1988].

*Analyze* the use of UML analysis models
*for the purpose of* understanding their utility
*with respect to* the comprehensibility and modifiability of source code
*from the point of view of* the maintainer
*in the context of* students in Computer Science and practitioners.

The GQM formalism ensures that important aspects are defined before the planning and the execution took place [Wohlin et al. 2000].

### 2.2. Context Selection

We used two systems in the family of experiments.

—*S1*. A software system to sell and manage CDs/DVDs in a music shop
—*S2*. A software system to book and buy theater tickets

Both systems were desktop applications based on the Model-View-Controller (MVC) architectural model. The documentation of these two systems was created within a course on Advanced Object Oriented Programming (AOOP) by its lecturer, who was not involved in the study presented here. The documentation (i.e., requirements analysis document, system design document, and object design document) was developed by adopting an incremental development process similar to the one suggested by Bruegge and Dutoit [2003]. One of the authors reviewed the documentation of the two systems to find possible issues. No remarkable modifications were needed to improve the documentation (e.g., typographical errors from the models were removed) and source code (e.g., the source code was indented).

The documentation of S1 and S2 was used by groups of 4 or 5 students to implement these software systems in Java as a laboratory activity of the AOOP course. In the software industry, the software engineers who design a system may be different from those that develop it. The documentation of S1 and S2 can be considered realistic enough for small-sized development projects of the following kind: in-house software (i.e., the system is developed inside the software company for its own use) or subcontracting (i.e., a subcontractor develops or delivers part of a system to a main contractor) [Lauesen 2002]. The students that developed these systems did not participate in the experiments. In these experiments, we used the source code that the lecturer of the AOOP course selected from among the software systems developed in the 2004–2005 and 2005–2006 academic years. We did not have any control on the selection process of these systems. However, we asked the lecturer to choose the implementation he considered the best for S1 and for S2, respectively. He opted for the implementations of the students who achieved the best grade in the AOOP course. For each experiment in the family, these design choices reduced both internal and external validity threats.

We selected two experimental objects within S1 and S2 keeping in mind a trade-off between complexity and relevance of the functionality chosen. For S1, we selected the feature *search for a singer*: the user inserts a string (e.g., the surname of the

singer), then the system searches for all the singers that satisfy the search criterion and shows them in a list with the associated information. For S2, we considered *buy a theater ticket*: the system shows the list of the available tickets for a given theater and performance, and then the user chooses the ticket and inserts data about the spectator. A functionality is represented with the successful use case. For both S1 and S2, exceptional and/or boundary conditions (e.g., if there is no singer that satisfies the search criteria specified by the user) were taken into account.

We selected analysis models and the associated source code so that a comprehension and modification task on them needed about one hour. The use of incomplete documentation and of a subset of the entire software system on which a maintenance operation impacts is quite common in the software industry. The documentation could be incomplete for several reasons. Examples are when only part of the documentation exists (e.g., in lean development processes), is up to date, or is useful to perform a maintenance operation (since it impacts only a few subsystems) [Bruegge and Dutoit 2003; McDermid 1991]. This is also the case when traceability management is exploited in a software development project [Asuncion et al. 2007; Lindvall and Sandahl 1996]. The experimental objects were selected to be similar to each other.

Analysis models accompanied chunks of source code implementing the specific functionality of S1 and S2 consisting of 463 and 378 LOCs (lines of code), respectively. The experimental object selected in S1 constituted 6 classes, while the other 5. We removed the comments within the source code of the two chunks of S1 and S2 to avoid biasing the results. The analysis models for S1 contained a use case diagram, two use cases described according to the template suggested by Bruegge and Dutoit [2003], a conceptual model (detailed information on each of the four selected classes was provided through a table that also included a short summary on the meaning and the role of the class), and two sequence diagrams (one was for the main success scenario, while the other was for a boundary condition). Similarly to S1, we gave for S2 a document with a use case diagram, two use cases, a conceptual model, and two sequence diagrams.

We conducted all the experiments in research laboratories under controlled conditions. For each experiment, the participants had the following characteristics.

—*E-UBAS*. The participants were students of a Software Engineering II course. During the bachelor program in Computer Science at the same university, the participants had passed all the exams related to the following courses: Software Engineering I, Object-Oriented Programming I and II, and Data Bases.
—*R1-UCLM*. The participants were students of a Software Engineering II course. The vast majority of the students had passed all the exams related to the following courses: Software Engineering I, Object-Oriented Programming I and II, and Data Bases.
—*R2-UCLM*. This replication was carried out as part of a Software Engineering II course. Most of the participants had passed the exams related to the Software Engineering I and Object-Oriented Programming I courses. The students of R1-UCLM and R2-UCLM were enrolled in different curricula. The modeling experience of these participants was less than those of R1-UCLM.
—*R3-UCLM*. This replication was performed with practitioners contracted in research projects in the Alarcos Research Group at the University of Castilla-La Mancha. Half of them had started working approximately a month before the replication took place, while the rest had between 5 and 8 months work of experience. The modeling experience of these participants was higher than that of participants in the other experiments. We asked the practitioners to participate in the experiment as part of their work hours to encourage them to participate.

The students participated in the experiments on a voluntary basis and were not paid.

### 2.3. Variable Selection

We considered students who were given source code without comments and without UML analysis models as comprising the *control group*, while the *treatment group* comprised students who were given source code (without comment) with UML analysis models. Thus, *method* is the independent variable (the main factor from here on). It is a nominal variable that can assume the following two values: AM (analysis models plus source code) and SC (source code alone).

The selected dependent variables are as follows.

—*Comp_Level*. This denotes the comprehension level of the source code achieved by a software engineer.
—*Modi_Level*. This denotes the capability of a maintainer to modify source code.

We used two questionnaires to obtain a quantitative evaluation of Comp_Level and Modi_Level, respectively. The questionnaires were composed of questions, each of which demanded an open answer. A sample question (here translated into English from Italian) of the comprehension task on S2 is. *In case the user selects a ticket for an already booked chair, what method is invoked and what message is shown to the user?* On the other hand, a sample question of the modification task on S2 is. *Which methods have to be modified to add an entry "Help" to the Start Menu and to change the theater name?* The complete list of questions of the comprehension and modification tasks for S1 and S2 are reported in the Appendix.

We used an information-retrieval-based approach [Baeza-Yates and Ribeiro-Neto 1999] to quantitively assess the answers obtained. Each answer is provided as string items (e.g., a sequence of method/class names and/or the text messages shown to a user), which are compared in turn with the expected items. Minor spelling issues in the string items are not considered as mistakes (e.g., $actionPerformed()$ vs. $actionPerform()$). The correctness of the obtained answers was measured with the precision measure, while the completeness was measured with the recall measure. These measures are defined as follows.

$$recall_{s,i} = \frac{|answers_{s,i} \cap correct_i|}{|correct_i|}, \tag{1}$$

$$precision_{s,i} = \frac{|answers_{s,i} \cap correct_i|}{|answers_{s,i}|}, \tag{2}$$

where $answers_{s,i}$ is the list of string items provided as the answer to question $i$ by participant $s$, and $correct_i$ is the correct list of string items expected for question $i$. In order to obtain a balance between correctness and completeness of an answer, we computed the harmonic mean of precision and recall (i.e., F-Measure).

To obtain a single measure representing the quality of the responses to all the questions of a questionnaire, we computed the overall average of the F-measure values. Both Comp_Level and Modi_Level are ratio scale measures.

We also analyzed the effect of the other independent variables (also called co-factors, from here on).

—*System*. This factor indicates the system (i.e., S1 or S2) used as the experimental object. The effect of the system factor should not be confounded with that of the main factor. Therefore, we selected well-known domains and experimental objects with a similar size and complexity.

Table II. Experiment Design

| Run | Group 1 | Group 2 | Group 3 | Group 4 |
|---|---|---|---|---|
| First | S1, AM | S1, SC | S2, AM | S2, SC |
| Second | S2, SC | S2, AM | S1, SC | S1, AM |

—*Run*. The participants were asked to accomplish two tasks in two subsequent laboratory runs (or trials). We analyzed whether passing from the first laboratory run to the next might affect the results.
—*Ability*. Students from the University of Basilicata with average marks[2] below 24/30 were classified as low-ability participants; otherwise, students were classified as high. In the replications conducted with students and professionals in Spain, the threshold was 9/10 of the exams passed. Participants with average marks below 9/10 were therefore classified as low; otherwise, high. For the professional, average mark was determined from the passed exams in their academic career. We used different threshold values in the experiments conducted in Italy and Spain because different grading systems are used in these countries. Our approach is similar to that proposed by Ricca et al. [2010] and by Abrahão et al. [2012].

### 2.4. Hypotheses Formulation

The following two null hypotheses have been formulated and tested.

*Hn0.* The Method (AM or SC) used does not significantly affect the participants' level of comprehensibility (Comp_Level) when performing source-code comprehension tasks.

*Hn1.* The Method (AM or SC) used does not significantly affect the participants' level of modifiability (Mod_Level) when performing source-code modification tasks.

The goal of the statistical analysis is to reject these null hypotheses and possibly to accept alternative ones (i.e., $Ha0 = \neg Hn0$ and $Ha1 = \neg Hn1$), which can easily be derived, because they admit a positive effect of Method. Both hypotheses are two sided because the results of our pilot study.

### 2.5. Design of the Experiment

In contrast to the pilot study, in E-UBAS and its replications, we used the within-participants counterbalanced experimental design (see Table II). This ensures that each participant worked on different experimental objects (S1 or S2) in two runs, using each of the methods once. This design was preferred to the original one, since it is particularly suitable for mitigating possible carry-over effects[3] and allows the effect of co-factors (e.g., system) to be studied. In addition, it represents the best choice when the number of participants is not so large.

All the experiments are balanced with respect to the number of participants assigned to AM and SC, and the assignment of participants to each group in Table II has been performed using ability as the blocking factor. Within E-UBAS, the number of participants in each group was 6 (3 students were high-ability participants). In this way, we had 48 observations in E-UBAS for each dependent variable: 24 for AM and 24 for SC. For AM and SC, 12 participants accomplished the task on S1 and 12 on S2.

[2]In Italy, exam marks are expressed as integers and assume values between 18 and 30. The lowest passing mark is 18, while the highest is 30. On the other hand, marks assume values between 5 and 10 in Spain. The lowest mark is 5, while the highest is 10.

[3]If a participant is tested first under condition A and then under condition B, he/she could potentially exhibit better or worse performances under condition B. In the first case, we talk of learning effect, while in the second case, of fatigue effect.

Table III. Post-Experiment Survey Questionnaire

| Id | Question | Possible Answers |
|---|---|---|
| **Q1** | I had enough time to perform the tasks | (1-5) |
| **Q2** | The task objectives were perfectly clear to me | (1-5) |
| **Q3** | The questions to be answered in the tasks were perfectly clear to me | (1-5) |
| **Q4** | Judge the difficulty of the tasks related to the system related to "search for a singer" | (A-E) |
| **Q5** | Judge the difficulty of the task related to the system related to buy a "theater ticket" | (A-E) |
| **Q6** | The analysis models were useful to comprehend the source code | (1-5) |
| **Q7** | The analysis models were useful to maintain the source code | (1-5) |
| 1 = strongly agree; 2 = agree; 3 = neutral; 4 = disagree; 5 = strongly disagree<br>A = very high; B = high; C = medium; D = low; E = very low | | |

As for R1-UCLM and R2-UCLM, we assigned 6 students to Group 1 and Group 2 each and 5 students to Group 3 and Group 4 each. The number of high-ability participants was the same in each group (i.e., 3) in R1-UCLM and R2-UCLM, respectively. Within R3-UCLM, we assigned 5 practitioners to Group 1 and Group 2 each (3 were high-ability participants). The other two groups contained 4 practitioners (2 were high-ability participants). The total number of observations in our family of experiments for each dependent variable can be easily computed by multiplying the number of participants by the number of runs (i.e., $86 \times 2 = 172$).

### 2.6. Experimental Tasks

We asked the participants to perform the following three tasks.

(1) *Comprehension task*. The participants were asked to fill in a questionnaire composed of 5 questions.
(2) *Modification task*. We also asked the participants to fill in a questionnaire to assess their capability in performing modification operations on source code. This questionnaire was composed of 4 questions. Note that the participants had to answer the questionnaire but did not have to carry out the real modifications of source code.
(3) *Post-experiment task*. At the end of the second run, we asked the participants to fill in the post-experiment questionnaire shown in Table III. The goal of this questionnaire was to obtain feedback about the participants' perceptions of the experiment execution and possibly to explain quantitative results. The answers to questions Q1, Q2, Q3, Q6, and Q7 were based on a five-point Likert scale [Oppenheim 1992] from strongly agree (1) to strongly disagree (5). Questions Q4 and Q5 demanded answers according to a different five-point Likert scale from very high (A) to very low (E).

### 2.7. Experiment Operation

The participants first attended an introductory lesson in which the experimenters presented detailed instructions on the experiment. Details on the experimental hypotheses were not provided, and the participants were informed that their grade on the course would not be affected by their performance (i.e., Comp_Level and Modi_Level). However, we rewarded the students for their participation with a bonus in their final mark. To familiarize then with the experimental procedure, the participants accomplished an exercise similar to that which would appear in the experimental tasks. The system used in that exercise was the *poker game*. The participants were provided with analysis models accompanied with a chunk of source code implementing the use case: *adding a new game*. We did not impose any time limit to accomplish that exercise. The participants were also informed that the data collected in the experiments were used for research purposes and treated confidentially.

After the introductory lesson, we assigned the participants to Group 1, Group 2, Group 3, and Group 4 (Table II). No interaction was permitted among the participants, both within each laboratory run and while passing from the first run to the second one. No time limit for performing each of the two runs was imposed.

To carry out the experiment, the participants first received the material for the first laboratory run, and when they had finished, the material for the second run was provided. After the completion of both runs, they were given the post-experiment questionnaire.

### 2.8. Analysis Procedure

To perform the data analysis, we carried out the following steps.

(1) We undertook the descriptive statistics of the measures of the dependent variables, that is, Comp_Level and Modi_Level.
(2) We planned to test the null hypotheses using unpaired analyses (i.e., each participant executed the two tasks on two different experimental objects). Then, we opted for the unpaired t-test in case data are normally distributed. The normality of the data is tested by the Shapiro-Wilk W test [Shapiro and Wilk 1965] (Shapiro test from here forward). The non-parametric Wilcoxon rank-sum test (also known as the Mann Whitney test) [Conover 1998] was the chosen alternative to the unpaired t-test.

    To strengthen the results of each experiment, we decided to integrate them using a meta-analysis. Meta-analysis is a set of statistical techniques for combining the different effect sizes of the experiments to obtain a global effect of a factor on a dependent variable (e.g., Method on Comp_Level and Method on Modi_Level). For each dependent variable, we computed the mean value obtained by the participants when using AM, minus the mean value they obtained with SC. We used these values to compute the Hedges' g metric [Hedges and Olkin 1985; Kampenes et al. 2007]. To obtain the overall conclusion, we calculated the Z score based on the mean and standard deviation of the Hedges' g statistics of the experiments. Therefore, the global effect size was obtained by using the Hedges' g metric, with the weights proportional to the experiment size:

$$\bar{Z} = \frac{\sum_i w_i z_i}{\sum_i w_i}, \tag{3}$$

    where $w_i = 1/(n_i - 3)$ and $n_i$ is the sample size of the $i$th experiment. The higher the value of Hedges' g, the higher the corresponding mean difference. An effect size of 0.5 indicates that the mean value obtained when using analysis models is half a standard deviation larger than the mean when not using them. As suggested by Kampenes et al. [2007], the effect size can be classified as small (S) for values between 0 and 0.37, medium (M) for values between 0.38 and 1.0, and large (L) for values above 1.00. The results of the meta-analysis are summarized by means of forest plots [Hedges and Olkin 1985].
(3) We also analyzed the influence of the co-factors. We planned to use a two-way Analysis of Variance (ANOVA) [Devore and Farnum 1999] if the data are normally distributed and if their variance is constant. We decided to use the tests of Shapiro and Wilk [1965] and Levene [1960] to verify these two assumptions, respectively. In the case that these assumptions were not verified, we opted for a two-way permutation test [Baker 1995], a non-parametric alternative to the two-way ANOVA.
(4) The responses to the post-experiment questionnaire were analyzed by using the median of the answers to each question. In addition, we verified whether the

participants consistently agreed with each statement of that questionnaire using statistical tests (i.e., the Mann Whitney test). We tested the null hypothesis that the responses are not significantly less than the mid-value (i.e., neutral or medium). This was possible because of the ordinal scales of the possible responses (i) from "strongly agree" to "strongly disagree" and (ii) from "very high" to "very low". Both these scales are encoded with integers from 1 to 5. Note that in this case, only non-parametric analyses are possible because the distribution of the mid-values is not normal.

In all the statistical tests performed, we decided (as customary) to accept a probability of 5% of committing a Type-1-Error [Wohlin et al. 2000] and used R (www.r-project.org) as the environment for statistical computing. To perform the meta-analysis, we used the Meta-Analysis v2 tool [Biostat 2006].

### 2.9. Differences between Pilot Study and the Family of Experiments

Our experience with the pilot study led us to make the following changes.

—The participants in all the experiments were more experienced than those in the pilot study. This alteration was made to better analyze the effect of more highly experienced participants on the comprehensibility and modifiability of the source code when performing a maintenance task.
—We modified the questionnaires used to assess the source-code comprehensibility and modifiability in the pilot study by making the questions open rather than closed. The rationale for this modification was based on the fact that open questions should, to as great an extent as possible, reduce the possibility of the participants guessing and hence giving correct answers by chance [Scanniello et al. 2011].
—We organized the questions into two groups: comprehensibility and modifiability. This change was made to analyze the effect of the analysis models on the comprehension of source code and on the participants' capacity to modify it. Therefore, two new dependent variables were introduced.
—We used a within-participant counterbalanced experimental design in all the experiments of the family. This alteration was made to better study the effect of Method and its interaction with the co-factors.
—We extended and modified the data analysis. In particular, the new dependent variables required the introduction of two new null hypotheses.
—A different group of experimenters conducted R1-UCLM, R2-UCLM, and R3-UCLM.

### 2.10. Documentation and Communication

Issues such as documentation [Shull et al. 2004] and communication among the experimenters [Vegas et al. 2006] may influence the success or the failure of replications. To handle these issues and to ensure consistency across the different experimenters, we used laboratory packages, knowledge-sharing mechanisms, and communication media. In particular, with regard to the documentation, a native speaker translated all the experimental material, which was initially written in Italian, into Spanish. The replicators supported the native speaker and helped him when needed (e.g., for translation of technical terms). Clarifications were asked of original experimenters when needed. The material to be translated included the post-experiment survey questionnaire, the comprehension and modification questionnaires, the data collection forms, and the software artifacts used in the experiments (i.e., analysis models and source code). We also shared (i) a document to provide a common background in order to reproduce the same experimental conditions in all the experiments and (ii) the paper in which the pilot study was presented [Gravino et al. 2010].

Table IV. Descriptive Statistics for Comp_Level

| | AM | | | SC | | |
|---|---|---|---|---|---|---|
| Exp_ID | Med. | Mean | Std. Dev. | Med. | Mean | Std. Dev. |
| E-UBAS | 0.765 | 0.728 | 0.155 | 0.725 | 0.731 | 0.132 |
| R1-UCLM | 0.580 | 0.556 | 0.233 | 0.625 | 0.625 | 0.234 |
| R2-UCLM | 0.255 | 0.332 | 0.256 | 0.585 | 0.586 | 0.200 |
| R3-UCLM | 0.475 | 0.479 | 0.229 | 0.625 | 0.606 | 0.225 |

Table V. Descriptive Statistics for Modi_Level

| | AM | | | SC | | |
|---|---|---|---|---|---|---|
| Exp_ID | Med. | Mean | Std. Dev. | Med. | Mean | Std. Dev. |
| E-UBAS | 0.81 | 0.77 | 0.203 | 0.71 | 0.709 | 0.152 |
| R1-UCLM | 0.52 | 0.531 | 0.23 | 0.565 | 0.558 | 0.154 |
| R2-UCLM | 0.3 | 0.376 | 0.243 | 0.44 | 0.449 | 0.26 |
| R3-UCLM | 0.48 | 0.474 | 0.215 | 0.56 | 0.517 | 0.266 |

We began with an initial face-to-face meeting in which the main ideas of the experiments were discussed and reported in minutes. We exchanged the minutes of this meeting by email in order to agree to a shared common research plan. This phase was relevant to sharing knowledge among the experimenters and to discussing possible issues related to the study.

We used instant messaging tools and emails to establish a communication channel in all the phases of the study (including the execution of the laboratory runs). We also executed teleconferences to share knowledge among the research groups and to discuss the experimental procedure to be used in the external replications. The results of the interactions were reported in a common document where all the decisions were recorded. This also reduced consistency issues across the experimenters.

## 3. RESULTS

In this section, we present the data analysis following the procedure previously presented.

### 3.1. Descriptive Statistics and Exploratory Analysis

Table IV and Table V show the descriptive statistics of Comp_Level and Modi_Level, respectively (i.e., median, mean, and standard deviation), grouped by Method.

—*Comp_Level*. On average, the participants achieved slightly better results when employing source code alone. Better median values for SC were achieved in all the experiments, except for E-UBAS. The difference in favor of SC is more evident for R2-UCLM and R3-UCLM.

—*Modi_Level*. In the three replications (i.e., R1-UCLM, R2-UCLM, and R3-UCLM), the participants achieved better results when using source code alone (see mean and median values). However, for E-UABS, the modifiability levels obtained by those participants employing UML analysis models was higher than the level obtained by the participants using source code alone.

We can observe that for Comp_Level, there is a clear tendency in favor of using source code alone (see the mean values). For Modi_Level, the results are less evident, and in three out of the four experiments, the participants obtained better Modi_Level when using source code alone. On both the dependent variables, the worst results were achieved in R3-UCLM. This difference with respect to the former experiments could be due to the participants' experience in programming and software modeling.

Table VI. Unpaired t-Test Results for Comp_level and Modi_level

| Exp_ID | Dependent Variable | #obs | p-value | Statistical Power | $\beta$-value | # of AM > SC | # of AM < SC | # of AM = SC |
|---|---|---|---|---|---|---|---|---|
| E-UBAS | Comp_Level | 48 | No (0.944) | 0.031 | 0.969 | 12/24 | 11/24 | 1/24 |
| | Modi_Level | 48 | No (0.09) | 0.197 | 0.803 | 15/24 | 8/24 | 1/24 |
| R1-UCLM | Comp_Level | 44 | No (0.335) | 0.225 | 0.775 | 9/22 | 12/22 | 1/22 |
| | Modi_Level | 44 | No (0.654) | 0.064 | 0.936 | 10/22 | 12/22 | 0/22 |
| R2-UCLM | Comp_Level | 44 | Yes (0.001) | 0.893 | 0.107 | 5/22 | 17/22 | 0/22 |
| | Modi_Level | 44 | No (0.344) | 0.131 | 0.869 | 11/22 | 11/22 | 0/22 |
| R3-UCLM | Comp_Level | 36 | No (0.105) | 0.327 | 0.673 | 8/18 | 10/18 | 0/18 |
| | Modi_Level | 36 | No (0.594) | 0.068 | 0.932 | 8/18 | 9/18 | 1/18 |

### 3.2. Influence of Method

*3.2.1. Testing Hn0.* For all the experiments, the Shapiro test returned p-values greater than 0.05, and so we used the unpaired t-test for Hn0. The results shown in Table VI indicated that there was no statistically significant difference when the participants did or did not employ analysis models to perform a comprehension task. This holds for all the experiments in the family with the exception of R2-UCLM, where a significant difference in favor of SC for Comp_Level was observed (p-value = 0.001; 95% confidence interval, $-0.395$ to $-0.115$). In this case, the value of the statistical power[4] was 0.893. The $\beta$-values are always high when the null hypotheses have not been rejected. The highest value was obtained for E-UBAS (0.969), while the lowest for R3-UCLM (0.673).

Table VI also shows the number of participants that achieved better Comp_Level values when using UML analysis models and source code together (# of AM > SC); and the source code alone (# of AM < SC). The number of participants that obtained the same Comp_Level values using AM and SC (# of AM = SC) is also shown. For all the experiments, with the exception of E-UBAS, the number of participants achieving better results with source code alone was greater.

*3.2.2. Testing Hn1.* We used the unpaired t-test because the Shapiro test returned p-values greater than 0.05 in all the experiments. The null hypothesis Hn1 cannot be rejected for Modi_Level (see Table VI). As with Comp_Level, we analyzed the number of participants that achieved better/worse values for Modi_Level with AM or SC. For E-UBAS, the number of participants who obtained better scores with AM was greater than the number of participants who achieved better scores using SC. For the three experiments in Spain, the number of participants who achieved better scores with SC was greater than the number of participants who obtained better scores with AM.

The results suggest that there was no statistically significant difference on Modi_Level when the participants did or did not employ UML analysis models. Nonetheless, the $\beta$-values are always greater than 0.8.

*3.2.3. Integrating the Obtained Results through Meta-Analysis.* Figures 1 and 2 show the forest plots for Comp_level and Modi_Level, respectively. The squares indicate the individual effect size of each experiment and the diamond (on the bottom) shows the global effect size. The squares and diamonds are proportional in size to each study's weight under the fixed effect model (see the "Relative weight" column). The figures also show the values of both the Hedges' g metric and the global effect size. Positive values of

[4]Statistical power is the probability that a test will reject a null hypothesis when it is actually false. The highest value is 1, while 0 is the lowest. The value 0.80 is considered as a standard for the adequacy [Ellis 2010]. That power is computed as 1 minus the Type 2 error rate (i.e., $\beta$-value). This kind of error rate is used to estimate the probability of accepting the null hypothesis when it is false. Therefore, when null hypotheses cannot be rejected, it is meaningful to consider $\beta$-values in the discussion of the results.

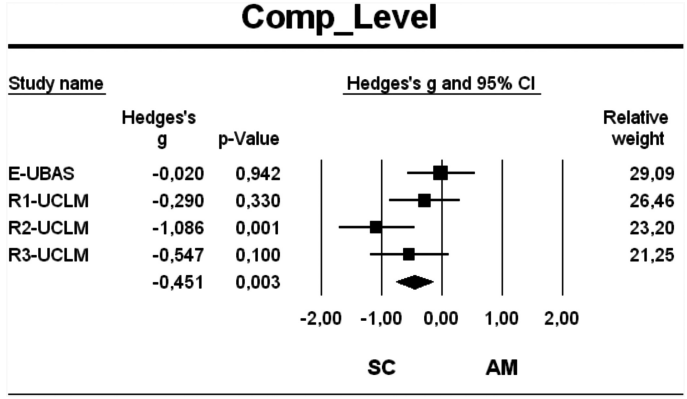


Fig. 1. Meta-analysis of Comp_Level.

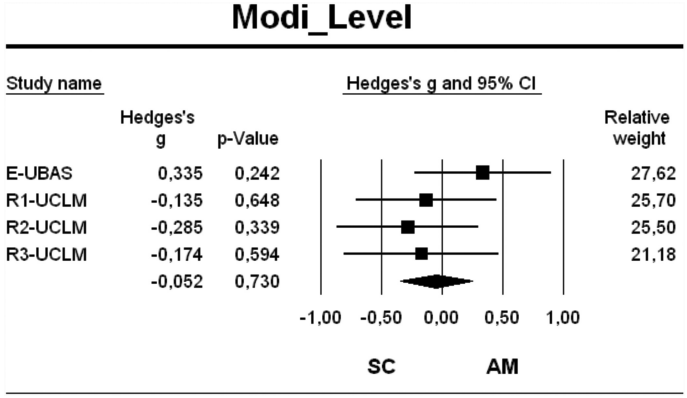


Fig. 2. Meta-analysis of Modi_Level.

the Hedges' g metric indicate that the use of analysis models improves the comprehensibility and modifiability of source code, while negative values signify that source code alone is the improving treatment. This implies that the models have a negative effect on Comp_Level and Modi_Level. In all the cases, with the exception of E-UBAS on Modi_Level, the participants achieved better values in both the dependent variables when using source code alone.

The global effect size was statistically significant only for Comp_Level (see Figure 1). The value obtained for the Hedge's g metric ($-0.451$) indicates a medium size for the global effect. The negative value reveals that the participants' level of comprehensibility is better when using source code alone. This effect is statistically significant and has a large effect size for R2-UCLM.

With regard to Modi_Level, most of the squares in Figure 2 are on the left-hand side, thus showing a tendency in favor of the use of SC. However, despite this tendency, the global effect size is not statistically significant.

### 3.3. Analysis of Co-Factors

The results of the analysis of the co-factors is summarized in Table VII. For each experiment, this table reports whether or not a co-factor has any effect on each of the two dependent variables. The obtained p-values are shown in brackets and are

Table VII. Analysis on the Co-Factors for Comp_Level and Modi_Level

| Exp_ID | Dependent variable | System | Run | Ability |
|---|---|---|---|---|
| E-UBAS | Comp_Level | No (0.852) | No (0.551) | No (0.425) |
| | Modi_Level (*) | Yes (0.014) | No (0.623) | No (0.372) |
| R1-UCLM | Comp_Level | Yes (<0.001) | No (0.214) | No (0.372) |
| | Modi_Level | No (0.125) | No (0.137) | Yes (0.021) |
| R2-UCLM | Comp_Level | Yes (0.041) | Yes (0.009) | No (0.145) |
| | Modi_Level | Yes (0.005) | Yes (0.034) | No (0.345) |
| R3-UCLM | Comp_Level | Yes (<0.001) | No (0.584) | No (0.122) |
| | Modi_Level | Yes (0.025) | Yes (0.025) | No (0.061) |

(*) These results were obtained using a two-way permutation test.

obtained with a two-way ANOVA. We could apply a two-way ANOVA in all the cases with the only exception of E-UBAS, where we applied a two-way permutation test. The normality assumption was not verified for E-UBAS, Modi_Level, and AM. In fact, the p-value returned by the Shapiro test was 0.008. We report the results about the interaction between Method and Run only for R2-UCLM on Comp_Level: in all the other cases, there was not a statistically significant interaction between Method and the co-factors on both the dependent variables.

*3.3.1. System.* We now discuss the results of the analysis of the co-factor System for the two dependent variables Comp_Level and Modi_Level.

*Comp_Level.* The results show that the effect of System on Comp_Level was not statistically significant for E-UBAS, while it was statistically significant for the three replications (p-values are <0.001, 0.041, and <0.001, respectively). The results suggested that the participants in the replications obtained better Comp_Level values when performing the task on S1, both using SC and AM.

*Modi_Level.* There was a positive effect of System in all the experiments with the exception of R1-UCLM. The p-values are 0.014, 0.005, and 0.025, respectively. The source code of S1 seemed to be more difficult to modify than the source code of S2. This result is in contrast with the one achieved on Comp_Level.

*3.3.2. Run.* We now show the results of the analysis conducted for Run on the two dependent variables.

*Comp_Level.* The results indicated that in all the experiments (with the exception of R2-UCLM), the effect of the co-factor Run was not statistically significant. This signifies that the participants in the second run did not obtain significantly larger or smaller results when using AM and SC on Comp_Level. With regard to R2-UCLM, the participants in the second run obtained significantly better results (p-value = 0.009). An interaction between Method and Run was also present (p-value = 0.004).

*Modi_Level.* The effect of Run was not statistically significant for E-UBAS and R1-UCLM. For R2-UCLM and R3-UCLM, the effect of Run was statistically significant (p-values are 0.034 and 0.025, respectively). The participants in each of these experiment obtained better Modi_Level values in the second run.

*3.3.3. Ability.* For the factor Ability, the results for the dependent variables are now reported.

*Comp_Level.* In all the experiments, the effect of Ability was not statistically significant on Comp_Level. This signifies that high and low participants did not obtain significantly larger or smaller differences on Comp_Level.

Table VIII. Post-Questionnaire Answers

| | E-UBAS | | R1-UCLM | | R2-UCLM | | R3-UCLM | |
|---|---|---|---|---|---|---|---|---|
| Quest. | median | p-value | median | p-value | median | p-value | median | p-value |
| Q1 | 1 | YES (<0.001 ) | 2 | YES (<0.001) | 1 | YES (<0.001) | 2 | YES (<0.001) |
| Q2 | 1 | YES (<0.001) | 2 | YES (0.002) | 2 | YES (<0.001) | 2 | YES (<0.001) |
| Q3 | 2 | YES (<0.001) | 2.5 | NO (0.07) | 3 | NO (0.117) | 2 | NO (0.117) |
| Q4 | 3 | YES (0.026) | 3 | YES (<0.001) | 2 | YES (0.001) | 2 | YES (0.001) |
| Q5 | 3 | NO (0.629) | 3 | YES (0.034) | 3 | NO (0.056) | 3 | NO (0.056) |
| Q6 | 2 | YES (<0.001) | 2 | YES (<0.001) | 2 | YES (<0.001) | 1 | YES (<0.001) |
| Q7 | 2.5 | NO (0.08) | 3 | NO (0.284) | 2 | YES (<0.001) | 2 | YES (<0.001) |

*Modi_Level.* The effect of Ability was not statistically significant on Modi_Level for E-UBAS, R2-UCLM, and R3-UCLM. The effect of Ability was statistically significant for R1-UCLM (p-value = 0.021). High ability participants achieved significantly better results than low ability ones on Modi_Level.

### 3.4. Post-Experiment Survey Questionnaire Results

Table VIII reports the median values of the answers to the post-experiment questionnaire grouped by experiment together with the p-values returned by the Mann Whitney test. The analysis of the responses revealed that the time needed to carry out the comprehension and modification tasks was considered appropriate, and the objectives of the tasks were clear for all the experiments. In particular, for all the experiments, the medians of the answers for Q1 and Q2 were 1 (strongly agree) and 2 (agree), respectively, and the p-values are less than 0.05. This signifies that the responses are significantly less than 3 (neutral), namely, the greater part of the participants answered "strongly agree" or "agree".

The questions to be answered in each task were generally considered to be clear in all the experiments, because the medians for Q3 were in between 2 (agree) and 3 (neutral), extremes included. For E-UBAS, the p-value is less than 0.001. In all the other experiments, we could not reject the null hypothesis that the responses are not significantly less than 3 (neutral).

The analysis of the answers to Q4 and Q5 indicated that the participants found the difficulty of the comprehension and modification tasks to be either medium or high (i.e., the values of the medians were 2 or 3). This finding is confirmed by the quantitative analysis on the responses. As for Q4, we could reject the null hypothesis that the responses are not significantly less than 3 (medium). In contrast, the null hypothesis for Q5 was not rejected (i.e., the participants found the tasks on S2 harder to perform). This held for all the experiments with the only exception being R1-UCLM.

Finally, the participants generally considered UML analysis models to be useful when performing the comprehension and the modification tasks. The medians of the answers to Q6 ranged from 1 (strongly agree) to 2 (agree), and the p-values are always less than 0.001. The medians of the answers to Q7 ranged from 2 (agree) to 3 (neutral). As for R2-UCLM and R2-UCLM, we rejected the null hypothesis that the responses are not significantly less than 3 (neutral). For E-UBAS and R1-UCLM, the hypothesis was not rejected.

## 4. DISCUSSION

Although we were not able to reject the null hypotheses defined in each experiment, the meta-analysis highlighted that the participants obtained better scores for Comp_Level when using SC. Therefore, we can conclude that analysis models did not help the participants to comprehend source code, although these models provided additional

information on the functionality implemented. This result might be because the models did not refer to objects of the solution domain, but to objects (or entities) of the problem domain. Even though the meta-analysis improves the findings for the individual studies, we cannot provide conclusive findings on whether analysis models helped in comprehending source code in the context of graduate and novice practitioners with respect to systems related to well-known domains and when only a part of the whole documentation and code is available.

With regard to Modi_Level, the meta-analysis did not allow us to obtain definitive results. However, the descriptive statistics reported in Table V showed a slight tendency in favor of SC. It might be possible that the participants did not adequately pay attention to the source code because they were distracted by the the documentation and the analysis models. We can postulate that UML analysis models referred to objects of the problem domain and then did not provide any useful information on the implementation.

It is also worth mentioning that the results of the family of experiments differ from those obtained in the pilot study [Gravino et al. 2010], in which we observed a slight tendency in favor of UML analysis models. This difference might be owing to the various improvements made (i.e., design and material) in all the experiments.

Regarding Ability, high-ability participants achieved better results than low-ability ones. For all the experiments, the comprehension achieved by high-ability participants was better than that achieved by low-ability participants when using AM. This result indicates that a particular ability is needed in order to not have analysis models affecting the participants' comprehension in case of undergraduate and graduate students and novice practitioners. To better analyze these differences, we computed the mean percentage differences.[5] For example, the differences of Comp_Level between high- and low-ability participants ranged from 1% for E-UBAS to 55% for R2-UCLM. As for Modi_Level, high-ability participants achieved higher scores than low-ability ones. The mean percentage differences on AM for Modi_Level ranged from 6% for R2-UCLM to 61% for R1-UCLM.

As for Ability and SC, the high-ability participants got higher Comp_Level values than low-ability ones: the mean percentage differences ranged from 9% for E-UBAS and R3-UCLM to 14% for R1-UCLM. The mean percentage differences ranged from 1% for E-UBAS to 46% for R3-UCLM on Modi_Level.

We also observed that for the three replications performed in Spain, S2 seemed to be more difficult than S1 in terms of comprehensibility, while the same participants achieved better results when performing the modification task on S2. These results did not allow us to provide a definitive conclusion about the influence of the co-factor System (i.e., whether S1 was more/less difficult than S2). This result could be justified by the participants' varying levels of familiarity with the problem and solution domains of the systems S1 and S2. This finding suggests that in the selected context, the familiarity with the problem domain could affect comprehensibility and modifiability more than the presence or the absence of analysis models. This point deserves specially conceived future investigations.

### 4.1. Implications of the Study

We adopted a perspective-based approach [Basili et al. 1996] to judge the implications of our family of experiments. In particular, we based our discussion on the *practitioner/consultant* (simply *practitioner* in the remainder at the article) and *researcher* perspectives [Kitchenham et al. 2008]. The main practical implications of our study can be summarized as follows.

[5]Given two values $(a, b)$, the mean percentage difference of $a$ and $b$ is computed as $(a - b)/b * 100$.

—The use of UML analysis models seems useless in the performance of maintenance operations. This result is relevant from both the practitioner and the researcher perspectives. From the practitioner perspective, this result is relevant because it could be useless to give additional information to maintainers when the individual maintainer performs small maintenance operations (e.g., corrective). From the researcher perspective, it is interesting to investigate whether variations in the context (e.g., larger systems and more or less experienced maintainers) might lead to different results and why analysis models could be not useful. Taking into consideration the results by Arisholm et al. [2006] (see the related work section) and those presented in this article, it could be possible that combining UML analysis models with models produced in the later phases of the development process (i.e., design models) improves comprehensibility and modifiability of source code. In fact, analysis models provide information on the software system from the perspective of the functionality to be implemented, while design models give details on the implementation. Analysis models and design models have different objectives that somehow complement each other. Although this perhaps might be not surprising, this study poses the basis of future investigations on how the combination of analysis and design UML models supports software engineers in the maintenance phase.
—UML analysis models seem to distract the participants while performing comprehension and modification tasks. This result is relevant for the researcher because it is interesting to investigate why participants (independently from the experience) do not get an improved comprehension of source code when it is combined with analysis models. A plausible justification for this result is that entity names have changed between the models and the code, and relationships between the models and the code have changed to become so much more intricate than what the models predicted. Again, combining analysis models with design models could make the difference [Scanniello et al. 2012].
—High-ability participants benefit more from UML analysis models than low-ability ones. This result is relevant for practitioners and researchers. Although the difference between high and low participants is not always statistically significant (see Section 3.3.3 and Section 4), they achieved on average better values for Comp_Level and Modi_Level when using AM. A possible justification for this result is that the UML is just a notation and then provides a weak support for semantics [Booch et al. 2005]. For example, in the context of class diagrams, a conceptual model mostly shows relationships between some entities, but the rationale behind those relationships (rooted in the domain rich information) or the meaning of those relationships is not conveyed. Then, it could be possible that high-ability participants are more proficient than low-ability ones in inferring and/or deducing the rationale/meaning of those relationships and the semantics behind these models.
—The study is focused on desktop applications for selling CDs/DVDs in a music shop and booking theater tickets. The documentation of these applications could be considered as developed in the following kinds of projects: in-house software or subcontracting [Lauesen 2002]. The researcher and the practitioner could be interested in answering the following question: Do the results observed hold for different kinds of software systems developed in different kind of projects? Our study represents the first step in this concern.
—Although we are not sure that our findings scale to real projects, the obtained results could be true in all the cases in which the documentation is incomplete (e.g., in lean development processes) and the maintenance operation is executed on a subset of the source code of the entire system.

—Since in 6 out of 8 cases, the effect of System is statistically significant, it seems that the familiarity with the problem domain of a software system has more effect on comprehensibility and modifiability of source code than the presence or the absence of UML analysis models (see Sections 3.2.1, 3.2.2, and 3.3.1). This result could be of interest for both researchers and practitioners. Both of them could be interested in investigating if the previous expertise of a maintainer with the domain could positively or negatively affect comprehensibility and modifiability of source code when completed with UML analysis models. Our justification of the familiarity with the problem domain cannot be completely concluded from our results, and so further investigations are needed.
—The achieved results (see Section 3.3.1) also suggest that comprehensibility and modifiability are not directly related: the source code of S1 was easier to comprehend and more difficult to modify, while the source code of S2 was easier to modify and more difficult to comprehend. The study presented here poses the basis of future investigations in the direction of investigating the relationships between comprehensibility and modifiability of source code. Therefore, this result is interesting from the researcher and practitioner perspectives.
—UML is widely used in the software industry [Dobing and Parsons 2006; Scanniello et al. 2010]. The achieved results are then useful for all the companies that exploit that notation as a support for software maintainers/developers to execute maintenance operations. Nowadays, studies on UML are required to understand the cases in which its use improves comprehensibility and maintainability of source code. There are only a few evaluations, as we will discuss in the related work section.

## 5. THREATS TO VALIDITY

### 5.1. Internal Validity

Internal validity threats are diminished by the design of the experiments we adopted. Each group of participants involved in the experiments worked on two different tasks, with and without analysis models. Nevertheless, there is still the risk that the participants might have learned how to improve their performances (i.e., comprehensibility and modifiability values) when passing from the first laboratory run to the second one. In all the experiments, the scores achieved by the participants were not significantly better in the second run (except for R2-UCLM on Comp_Level and Modi_Level and for R3-UCLM on Modi_Level).

Another possible threat to external validity concerns the fact that no time limit was imposed to perform the tasks. It could be possible that the experiments were not able to reveal differences because the participants had enough time to answer the questions on the comprehension and modification questionnaires. We opted for this design choice because this is quite common in experiments similar to ours [Sjøberg et al. 2005] and because most experienced participants could have difficulty performing tasks under a time limit [Mendonça et al. 2008].

For each experiment, the internal validity threat was also mitigated because the participants had a similar amount of experience with the UML, software system modeling, and computer programming. Furthermore, all the participants found the material provided, the tasks, and the goals of the experiment to be clear, as the post-experiment survey questionnaire results showed.

Another issue concerns the exchange of information among the participants. The participants were not allowed to communicate with each other. We prevented this by monitoring them both during the runs and during the break between the two laboratory runs. When the experiment was concluded, the participants were asked to give back all the experimental material.

### 5.2. External Validity

External validity may be threatened when experiments are performed with students, thus leading to doubts concerning the representativeness of the participants with regard to software professionals. However, the tasks to be performed did not require a high level of industrial experience, so we believe that the use of students as participants could be considered appropriate, as suggested in literature [Carver et al. 2003; Höst et al. 2000]. Working with students also implies various advantages, such as the fact that the students' prior knowledge is rather homogeneous, there is the possible availability of a large number of participants [Verelst 2004], and there is the chance to test experimental design and initial hypotheses [Sjøberg et al. 2005]. An additional advantage of using students is that the cognitive complexity of the objects under study is not hidden by participants' experience. Nonetheless, in order to strengthen the external validity, we replicated the original experiment using 18 practitioners (R3-UCLM). Although the number of practitioners is not so large, it is compatible with that of other similar empirical investigations (e.g., [Abrahão et al. 2012; Arisholm et al. 2006]). It was hard for us to find a larger number of practitioners because of their availability and the restrictions of the research projects where we recruited the participants involved.

Another threat to external validity concerns the experimental objects used. The original experimenters, and then the external replications, were not involved in the realization of the software documentation used and in the implementation of the system used in the experiments. The size of the experimental objects could also threaten the external validity of the results. The rationale for selecting the used experimental objects relies on the need to simulate actual comprehension tasks related to small maintenance operations that novice software engineers and/or junior programmers may perform in a software company. Larger experimental objects could excessively overload the participants, thus biasing the experiments and their results. Also, the use of the source-code printout to execute the tasks (both using SC and AM) could have threatened external validity: the participants could only statically analyze the source code. This design choice was taken because the effect of executing the systems to solve comprehensibility/modifiability tasks could be confounded with the effect of the main factor. To confirm the results, we are going to conduct case studies in real software development projects, with practitioners in their own projects and over a much longer period of time. Using both controlled experiments and industrial case studies will allow us to obtain a more credible body of knowledge on the effect of UML analysis models. Nevertheless, from a pragmatic perspective, controlled experiments allow for better understanding of issues and factors to be considered afterwards in the industrial case studies [Arisholm et al. 2006]. This is because we opted for a family of experiments and conducted it before industrial case studies.

### 5.3. Construct Validity

Construct validity may be influenced by the measures used to obtain a quantitative evaluation of comprehensibility and modifiability, the questionnaires to assess these concerns, the post-experiment survey questionnaire, and social threats. We used a well-known and widely used measure to obtain a quantitative evaluation of comprehensibility and modifiability (e.g., [Ricca et al. 2010; Scanniello et al. 2011]). One of the authors defined the questionnaires used to assess these aspects. Furthermore, the comprehension/modification questionnaires were formulated to condition their answers in favor of neither SC nor AM. The questions were also sufficiently complex without being too obvious and were formulated in a similar form.

In all the experiments of our family, we considered two experimental objects. Therefore, the construct is underrepresented: the tasks which are measured could fail to

include important dimensions or facets of the construct [Wohlin et al. 2000]. We deliberately varied only the experience of the participants and the environments because changing a larger number of factors among the experiments could have a negative effect on tracing the results onto the main factor [Shull et al. 2008].

We conducted external replications to mitigate construct validity. In order to reduce consistency issues across the different experimenters, we carefully managed communication among the experimenters (see Section 2.10).

Other possible threats to construct validity could be related to the translation of the experimental material and social threats. To reduce the first kind of threat, we involved a native speaker to translate all the material used. To avoid social threats (i.e., evaluation apprehension), we did not grade the students on the results obtained in the experiments.

To support and explain the quantitative results of the experiments, we used a post-experiment survey questionnaire. It was designed using standard approaches and scales [Oppenheim 1992].

### 5.4. Conclusion Validity

Conclusion validity threats concern the issues that affect the ability to draw a correct conclusion. We used statistical tests to reject the null hypotheses. In particular, we exploited parametric statistical tests when normality was verified, and non-parametric statistical tests otherwise. A power analysis has been also performed.

Regarding the selection of the populations, we drew fair samples and conducted our experiments with participants belonging to these samples. Another threat could be related to the number of participants. This threat has been mitigated by conducting our investigation with a large number of participants: 86 participants in the family (and 16 participants in the pilot study). Due to the experimental design, the number of observations in our family of experiments was 172 in total. The results of the original experiment were confirmed with stronger evidence in all the replications.

The reliability of the used measures is another possible threat to conclusion validity. The used measures allowed us to assess in an objective and repeatable way the concerns under study: comprehensibility and modifiability.

## 6. RELATED WORK

The benefits of software documentation for comprehending and modifying source code have been largely studied (e.g., [Abbes et al. 2011; Scanniello et al. 2010; de Souza et al. 2005; Tilley and Huang 2003; Tryggeseth 1997]). In this scenario, we present hereafter research work related to our study. For example, Tryggeseth [1997] reported an experiment carried out with 34 participants in Norway. The object under study was a system comprising of 2.7K lines of code and around 100 pages of documentation: mostly textual documentation including requirements specification, design documentation, a test report, and a user manual. These participants were asked to record the time they spent on different enhancement-maintenance tasks on that system. The following empirical findings were reported: (1) the aid of having documentation available during system maintenance reduces the time needed to understand the system and the changes implied by a change request, and (2) it also enables the maintainer with more time and better knowledge to implement more detailed changes.

With respect to the usefulness of documenting design pattern instances to comprehend source code, only a few studies have been reported. Prechelt et al. [2002] presented two experiments to investigate whether maintainers are better supported in the comprehension of source code when design pattern instances are or are not explicitly documented. The experiment was performed on Java source code by 74 German graduate students, while the replication was on C++ source code using 22 American

undergraduate students. The results revealed that maintenance tasks were completed faster and with fewer errors if design pattern instances were explicitly documented. Gravino et al. [2011] performed two experiments with Master students in Computer Science at two Italian universities. The 24 participants in the original experiment performed a comprehension task with and without graphically-documented (with UML class diagrams) design pattern instances. Design pattern instances textually documented in the source code (as comments) were provided or were not to the 17 participants to perform a comprehension task within the replication. The results of this empirical investigation provided evidence that maintainers achieved better comprehension of the source code when design pattern instances are graphically documented and provided as a complement to the source code. This kind of documentation has a statistically significant effect on the task completion time and on the efficiency to accomplish that task. The replication results suggested that the effect of textually documented design pattern instances was not statistically significant on source-code comprehension. However, descriptive statistics indicated a trend in favor of this kind of documentation. Subsequently, the same authors [Gravino et al. 2012] conducted a further replication with 25 professional software developers. In that replication, the participants were divided into three groups. Depending on the group, each participant was or was not provided with graphical (i.e., with UML class diagrams) or textual (i.e., source-code comments) representations of the design pattern instances within the source code. The results revealed that participants provided with the documentation of the design pattern instances (both textual and graphical) achieved a significantly better comprehension than the participants with source code alone. Summarizing, the results achieved in these three studies suggest that documentation is useful for software maintenance.

Regarding the use of the UML as part of the documentation of an object-oriented software system, two systematic literature reviews have recently been published. Budgen et al. [2011] studied empirical investigations on the widely used UML notations and their usefulness. Fernández-Saez et al. [2012] collected the existing literature focused on the quality of UML models. Both systematic literature reviews show that comprehensibility and modifiability are the major concerns. It was also shown that there are few evaluations on how UML models support software engineers in the whole software development life cycle. In particular, very few papers report the use of UML in the maintenance of source code. In particular, Dzidek et al. [2008] presented the results of two controlled experiments carried out with students from different universities. Unlike us, they considered UML documents, including a use case diagram, sequence diagrams for each use case, and class diagrams. The quantitative results showed that UML models did not make a significant impact on the time needed to perform the modification tasks, both excluding and including the time needed to update the documentation. The quality of the modifications was greater when the participants had UML models. The effect of participants' ability and experience is not analyzed. Arisholm et al. [2006] presented the results of a controlled experiment carried out to assess the impact of UML design models on software maintenance. Software professionals were involved. The authors analyzed the time taken to perform the modifications to the system, the time spent on maintaining the models, and the quality of the modifications performed. The results of the quantitative analysis revealed no significant difference in the time spent making the modifications. Similarly to Dzidek et al. [2008], they observed that the quality of the modifications was higher for those participants who were furnished with UML models. In some sense, our work fills in a gap in that work, explicitly considering models produced in a given phase of the development process: models produced in the requirement engineering process and design phase have been considered together [Arisholm et al. 2006]. Another difference with respect to our study is that the authors analyzed the effect of UML-based documentation (a use case

diagram, sequence diagrams for each use case, and a class diagram) on modification tasks performed both on UML diagrams and source code. Again, our work fills in a gap in that work by considering the effect of analysis models on the comprehensibility of source code. Similar to Dzidek et al. [2008], the participants' ability and experience were not analyzed with respect to comprehensibility and modifiability of source code.

The research work presented in this article is different from previous work, since it pursues a different goal. In particular, the focus here is on the UML models produced in the early phases of the development process: requirements elicitation and analysis. The results shown here, and those of the studies just discussed, suggest that software documentation is useful in comprehending and modifying source code only when it includes models that represent aspects of the solution domain and that provide information on the implementation of the system under maintenance.

## 7. CONCLUSION AND FUTURE WORK

Many software projects do not develop a complete set of models throughout the entire software development life cycle. This implies that in many projects, the only models that are available are those produced in the requirements analysis process [Anda et al. 2006], which could also provide an incomplete abstraction of the functionality to be implemented in the software system under development. Therefore, we decided to carry out a pilot study [Gravino et al. 2010] and a family of experiments to investigate whether the use of UML models produced in the requirements elicitation and analysis phases supports software engineers in comprehending and modifying source code.

The family consisted of four experiments carried out with students and practitioners from Italy and Spain. We used controlled experiments because a number of confounding and uncontrollable factors could be present in real project settings. In real projects, it may be impossible to control factors such as learning and/or fatigue effects and to select specific tasks. Controlled experiments also reduce failure risks related to long-term empirical investigations (as in our case). Although questions about the external validity (e.g., generalization to realistic comprehension tasks on object-oriented source code) may arise, controlled experiments are often conducted in the early steps of empirical investigations that take place over years (e.g., [Arisholm et al. 2006; Colosimo et al. 2009]).

The achieved results suggested that the UML analysis models seemed to not improve the comprehensibility and modifiability of source code. The results regarding modifiability are less conclusive, although there was a slight tendency towards confirming the results found as regards comprehensibility. The questionable utility of the UML in our experimental context might be caused by the kind of models used: they do not provide any information on the implementation, so they need to be combined with design models. These results are perhaps not overly surprising, but it is acceptable, as evidence needs to be verified/reaffirmed through empirical studies [Basili et al. 1999; Kitchenham et al. 2002; Shull et al. 2008].

Possible future directions for our research are (i) performing further experimentation considering different and larger software systems related to unknown domains to verify whether the findings obtained are still valid; (ii) studying the effect of providing the participants with information in an incremental manner; (iii) analyzing the effect of different UML notations (e.g., activity diagrams) and models (e.g., design models); and (iv) investigating the effect of the same UML diagrams as we used here on non-source-code comprehension tasks.

## APPENDIX

The questionnaires (here translated into English from Italian) for S1 and S2 are shown here. We first list the questions used to obtain a quantitative evaluation of Comp_Level

on S1, and then those to quantitatively asses Modi_Level. The appendix concludes by reporting the questions of S2 for Comp_Level first and then for Modi_Level.

**S1. Comp_Level**

(1) If the operator introduces the name of a singer who is not in the shop database, which method is executed and what message is shown? In the case that no name is introduced, what is the message shown?
(2) When the *Control* class is instanced, which object does the constructor class create (i.e., *Control()*)?
(3) Which class/es and method/s are in charge of initializing the possible actions?
(4) Which class containing the field is to be used to perform a search for a singer?
(5) Which class and method are in charge of handling exceptions with respect to *SearchBySinger*?

**S1. Modi_Level**

(1) Which kind of classes should you create to add a new functionality for searching a record by its identifier?
(2) Which method in which class should you modify to handle the exceptional condition that an author is present in the database but no albums are associated?
(3) Which class/es are to be modified in order to change the visualization of the results for *SearchBySinger*?
(4) How should the *Controller* class be modified in order to trace the exceptional conditions when loading an album list?

**S2. Comp_Level**

(1) In case the user selects a ticket for an already booked chair, what method is invoked and what message is shown to the user?
(2) Which class/es and method/s are in charge of initializing the graphical user interfaces?
(3) Which class/es and method/s are in charge of loading the list of tickets?
(4) Which class/es and method/s are in charge of displaying the available tickets that can be purchased?
(5) If the user does not select any ticket to purchase, which method of which class is invoked? What is the displayed message?

**S2. Modi_Level**

(1) Which class/es are in charge of handling the new exception conditions related to *TicketPurchase*?
(2) Which class/es and method/s are to be coded when a new graphical interface has to be added?
(3) Which methods have to be modified in order to add an entry "Help" to the *Start Menu* and to change the theater name?
(4) Which method/s in which class/es should be modified if you wanted to change all the error messages corresponding to the exception conditions of *TicketPurchase*?

## REFERENCES

Marwen Abbes, Foutse Khomh, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2011. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In *Proceedings of the European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, Los Alamitos, CA, 181–190.

Silvia Mara Abrahão, Carmine Gravino, Emilio Insfran Pelozo, Giuseppe Scanniello, and Genoveffa Tortora. 2012. Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: results from a family of five experiments. *IEEE Trans. Softw. Eng.*

Ritu Agarwal and Atish P. Sinha. 2003. Object-oriented modeling with UML: A study of developers' perceptions. *Commun. ACM* 46, 9, 248–256.

Bente Anda, Kai Hansen, Ingolf Gullesen, and Hanne Kristin Thorsen. 2006. Experiences from introducing UML-based development in a large safety-critical project. *Empirical Softw. Eng.* 11, 4, 555–581.

Erik Arisholm, Lionel C. Briand, Siw Elisabeth Hove, and Yvan Labiche. 2006. The impact of UML documentation on software maintenance: An experimental evaluation. *IEEE Trans. Softw. Eng.* 32, 365–381.

Hazeline U. Asuncion, Frédéric François, and Richard N. Taylor. 2007. An end-to-end industrial software traceability tool. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 115–124.

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

R. Baker. 1995. Modern permutation test software. In *Randomization Tests*, 3rd Ed., E. Edgington, Ed. Marcel Dekker, Inc., New York, NY.

Victor Basili, Forrest Shull, and Filippo Lanubile. 1999. Building knowledge through families of experiments. *IEEE Trans. Softw. Eng.* 25, 4, 456–473.

Victor R. Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Lars Sivert Sørumgård, and Marvin V. Zelkowitz. 1996. The empirical investigation of perspective-based reading. *Empirical Softw. Eng.* 1, 2, 133–164.

Victor R. Basili and H. Dieter Rombach. 1988. The TAME project: Towards improvement-oriented software environments. *IEEE Trans. Softw. Eng.* 14, 6, 758–773.

Biostat. 2006. Comprehensive Meta-Analysis v. 2. In *Biostat Manual*, Englewood, NJ.

Grady Booch, James Rumbaugh, and Ivar Jacobson. 2005. *Unified Modeling Language User Guide*, 2nd Ed. Addison-Wesley Professional.

B. Bruegge and A. H. Dutoit. 2003. *Object-Oriented Software Engineering: Using UML, Patterns and Java*, 2nd Ed. Prentice-Hall.

David Budgen, Andy J. Burn, O. Pearl Brereton, Barbara A. Kitchenham, and Rialette Pretorius. 2011. Empirical evidence about the UML: A systematic literature review. *Softw. Pract. Exp.* 41, 4, 363–392.

Jeffrey Carver, Letizia Jaccheri, Sandro Morasca, and Forrest Shull. 2003. Issues in using students in empirical studies in software engineering education. In *Proceedings of the International Symposium on Software Metrics*. IEEE Computer Society, 239.

David Cohen, Mikael Lindvall, and Patricia Costa. 2004. An introduction to agile methods. *Adv. Comput.* 62, 1–66.

M. Colosimo, A. De Lucia, G. Scanniello, and G. Tortora. 2009. Evaluating legacy system migration technologies through empirical studies. *Inf. Softw. Technol.* 51, 12, 433–447.

W. J. Conover. 1998. *Practical Nonparametric Statistics.* 3rd Ed. Wiley.

Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2005. A study of the documentation essential to software maintenance. In *Proceedings of the International Conference on Design of Communication: Documenting & Designing for Pervasive Information*. ACM, New York, NY, 68–75.

J. L. Devore and N. Farnum. 1999. *Applied Statistics for Engineers and Scientists*. Duxbury.

B. Dobing and J. Parsons. 2006. How UML is used. *Commun. ACM* 49, 5, 109–113.

Wojciech J. Dzidek, Erik Arisholm, and Lionel C. Briand. 2008. A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *IEEE Trans. Softw. Eng.* 34, 407–432.

P. Ellis. 2010. *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*. Cambridge University Press.

John Erickson and Keng Siau. 2007. Theoretical and practical complexity of modeling methods. *Commun. ACM* 50, 8, 46–51.

A. Fernández-Saez, M. Genero, J. Nelson, G. Poels, and M. Piattini. 2012. A systematic literature review on the quality of UML models. *J. Data. Manage.* 22, 3, 46–70.

Carmine Gravino, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. 2011. Does the documentation of design pattern instances impact on source code comprehension? Results from two controlled experiments. In *Proceeding of the Working Conference on Reverse Engineering*. IEEE Computer Society, Los Alamitos, CA, 67–76.

Carmine Gravino, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. 2012. Do professional developers benefit from design pattern documentation? A replication in the context of source code comprehension. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*. Springer-Verlag, Berlin, 185–201.

Carmine Gravino, Genoveffa Tortora, and Giuseppe Scanniello. 2010. An empirical investigation on the relation between analysis models and source code comprehension. In *Proceedings of the ACM Symposium on Applied Computing*. ACM, 2365–2366.

Martin Grossman, Jay E. Aronson, and Richard V. McCarthy. 2005. Does UML make the grade? Insights from the software development community. *Inf. Softw. Technol.* 47, 6, 383–397.

L. Hedges and I. Olkin. 1985. *Statistical Methods for Meta-Analysis*. Academia Press.

Martin Höst, Björn Regnell, and Claes Wohlin. 2000. Using students as subjects: Comparative study of students and professionals in lead-time impact assessment. *Empirical Softw. Eng.* 5, 3, 201–214.

Andreas Jedlitschka, Marcus Ciolkowski, and Dietmar Pfahl. 2008. Reporting experiments in software engineering. In *Guide to Advanced Empirical Software Engineering*, Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, Eds., Springer, 201–228.

N. Juristo and A. Moreno. 2001. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Englewood Cliffs, NJ.

Vigdis By Kampenes, Tore Dybå, Jo Erskine Hannay, and Dag I. K. Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.* 49, 11–12, 1073–1086.

Barbara Kitchenham, Hiyam Al-Khilidar, Muhammed Babar, Mike Berry, Karl Cox, Jacky Keung, Felicia Kurniawati, Mark Staples, He Zhang, and Liming Zhu. 2008. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Softw. Eng.* 13, 97–121.

B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.* 28, 8, 721–734.

Soren Lauesen. 2002. *Software Requirements: Styles and Techniques*. Addison-Wesley.

H. Levene. 1960. Robust tests for equality of variances. In *Contributions to Probability and Statistics*, I. Olkin Ed., Stanford University Press., Palo Alto, CA.

M. Lindvall and K. Sandahl. 1996. Practical implications of traceability. *Softw. Pract. Exp.* 26, 10, 1161–1180.

Mitch Lubars, Colin Potts, and Charlie Richter. 1993. A review of the state of the practice in requirements modeling. In *Proceedings of the International Symposium on Requirements Engineering*. IEEE Computer Society Press, 2–14.

J. McDermid. 1991. *Software Engineer's Reference Book*. Butterworth-Heinemann, Ltd., Oxford, U.K.

Manoel G. Mendonça, José C. Maldonado, Maria C. F. de Oliveira, Jeffrey Carver, Sandra C. P. F. Fabbri, Forrest Shull, Guilherme H. Travassos, Erika Nina Höhn, and Victor R. Basili. 2008. A framework for software engineering experimental replications. In *Proceedings of the International Conference on Engineering of Complex Computer Systems*. IEEE Computer Society, Los Alamitos, CA, 203–212.

OMG. 2010. Unified Modeling Language (TM) URL. Tech. Rep. Object Management Group. http://www.uml.org.

A. N. Oppenheim. 1992. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London.

Lutz Prechelt, Barbara Unger-Lamprecht, Michael Philippsen, and Walter F. Tichy. 2002. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE Trans. Softw. Eng.* 28, 6, 595–606.

Filippo Ricca, Massimiliano Di Penta, Marco Torchiano, Paolo Tonella, and Mariano Ceccato. 2010. How Developers' experience and ability influence web application comprehension tasks supported by UML stereotypes: a series of four experiments. *IEEE Trans. Softw. Eng.* 36, 1, 96–118.

Giuseppe Scanniello, Carmine Gravino, and Genny Tortora. 2010. Investigating the role of UML in the software modeling and maintenance—A preliminary industrial survey. In *Proceedings of the International Conference on Enterprise Information Systems*. SciTePress, 141–148.

Giuseppe Scanniello, Carmine Gravino, and Genoveffa Tortora. 2012. Does the combined use of class and sequence diagrams improve the source code comprehension?: Results from a controlled experiment. In *Proceedings of the International Workshop on Experiences and Empirical Studies in Software Modelling*. ACM, New York, NY, 25–30.

G. Scanniello, F. Ricca, and M. Torchiano. 2011. On the effectiveness of the UML object diagrams: A replicated experiment. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering*. IET Digital Library, 76–85.

S. Shapiro and M. Wilk. 1965. An analysis of variance test for normality. *Biometrika* 52, 3–4, 591–611.

Forrest Shull, Jeffrey C. Carver, Sira Vegas, and Natalia Juristo Juzgado. 2008. The role of replications in empirical software engineering. *Empirical Softw. Eng.* 13, 2, 211–218.

Forrest Shull, Manoel G. Mendoncça, Victor Basili, Jeffrey Carver, José C. Maldonado, Sandra Fabbri, Guilherme Horta Travassos, and Maria Cristina Ferreira. 2004. Knowledge-sharing issues in experimental software engineering. *Empirical Softw. Eng.* 9, 1–2, 111–137.

D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N. Liborg, and A. C. Rekdal. 2005. A survey of controlled experiments in software engineering. *IEEE Trans. Softw. Eng.* 31, 9, 733–753.

Scott Tilley and Shihong Huang. 2003. A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. In *Proceedings of the International Conference on Documentation*. ACM, New York, NY, 184–191.

Eirik Tryggeseth. 1997. Report from an experiment: Impact of documentation on maintenance. *Empirical Softw. Eng.* 2, 2, 201–207.

S. Vegas, N. Juristo, A. Moreno, M. Solari, and P. Letelier. 2006. Analysis of the influence of communication between researchers on experiment replication. In *Proceedings of the International Symposium on Empirical Software Engineering*. ACM, New York, NY, 28–37.

Jan Verelst. 2004. The influence of the level of abstraction on the evolvability of conceptual models of information systems. In *Proceedings of the International Symposium on Empirical Software Engineering*. IEEE Computer Society, 17–26.

Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. 1998. Scenarios in system development: Current practice. *IEEE Softw.* 15, 34–45.

C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. 2000. *Experimentation in Software Engineering-An Introduction*. Kluwer.

Received July 2012; revised February, May 2013; accepted June 2013