Source-Code Comprehension Tasks Supported by UML Design Models: Results from a Controlled Experiment and a Differentiated Replication

Carmine Gravino\textsuperscript{a,*}, Giuseppe Scanniello\textsuperscript{b}, Genoveffa Tortora\textsuperscript{a}

\textsuperscript{a}DISTRA-MIT, University of Salerno
\textsuperscript{b}DiMIE - University of Basilicata

Abstract

Objective: The main objective is to investigate whether the comprehension of object-oriented source-code increases when it is added with UML class and sequence diagrams produced in the software design phase.

Methods: We conducted a controlled experiment and a differentiated replication with young software maintainers. In particular, groups of Bachelor and Master students were involved.

Results: The results show that more experienced participants better comprehend source-code when added with UML design models. An average improvement (or benefit) of circa 12\% was achieved when the participants accomplished the comprehension task with UML class and sequence diagrams. The results of an analysis on the time to accomplish comprehension tasks showed that less experienced participants significantly spent more time when comprehending source-code with UML design models. This kind of participants spent on average 44.8\% of the time to accomplish the same task with source-code alone.

Implications: It is useless to give UML design models to comprehend source-code in case maintainers are not adequately experienced with the UML. Furthermore, the less the experience of participants, the more the time to accomplish a comprehension task with UML diagram is.

\textsuperscript{*}Corresponding author

Email addresses: gravino@unisa.it (Carmine Gravino),
giuseppe.scanniello@unibas.it (Giuseppe Scanniello), tortora@unisa.it (Genoveffa Tortora)

Keywords: Design Models, Controlled Experiment, Source-Code Comprehension.

1. Introduction

Several issues (e.g., technical and managerial) contribute to the cost to execute comprehension tasks and might affect the comprehension of source-code [1]. For example, the absence of software documentation might impact on both these aspects. Even when the documentation is present and adequate, source-code comprehension remains a time consuming activity because very often software maintainers have to comprehend source-code by reading documentation that include software models produced by other maintainers [2]. Then, a source-code comprehension task and its related cost may be strictly conditioned by: the maintainers’ experience on the notation used to build the models and on their ability to accomplish comprehension tasks.

Nowadays the documentation of object-oriented software systems contains several models built with the UML (Unified Modeling Language) [3]. The assessment of the benefits deriving from the use of the UML in all the phases of the software life cycle is relevant for the software engineering community, as shown by the number of empirical studies in terms of controlled experiments and case studies available in the literature [4, 5]. Although a number of studies have been conducted on the UML, only a few of them have been carried out to assess whether the benefits from the use of the UML (if any) are related to maintainers’ ability and experience (e.g., [6, 7]).

In this paper, we present the results of a controlled experiment and a differentiated replication\footnote{Differentiated replications introduce variations in essential aspects of the experimental conditions [8]. One prominent variation concerns the executions of replications with different kinds of participants. This kind of replication could be also named independent or conceptual replication [9].} to study the effect of maintainers’ experience on the comprehension of source-code when UML class and sequence diagrams produced in the software design phase are provided to them. The original experiment was conducted with final year Master students in Computer Science. The preliminary results of that investigation have been presented in [10]. Second year Bachelor students in Computer Science were involved in the differentiated replication. In the study presented here, we first analyze whether
source-code comprehension increases when young software maintainers are provided with source-code and UML class and sequence diagrams produced in the design phase. The time to complete a source-code comprehension task is also analyzed and the results have been presented and discussed.

Structure of the paper. In Section 2, we discuss related work. The design of the experiments is presented in Section 3, while we present and discuss the achieved results in Section 4. The threats to validity are highlighted in Section 5. Finally, we show possible future directions for our research.

2. Related Work

We first discuss the related literature concerning controlled experiments aimed at assessing the effect of using the UML in software maintenance and program comprehension. We conclude this section presenting related work on the influence of ability and experience on the execution of comprehension tasks supported by the UML. To get a deeper understanding of empirical evaluations on the models and forms used in the UML, a systematic literature review is available in [4].

2.1. Empirical studies on the UML

Arisholm et al. [11] observe that the availability of documentation based on the UML may significantly improve the functional correctness of changes as well as the design quality when complex tasks have to be accomplished. Although this study and the one presented here have the same research goal (i.e., studying the impact of UML based documentation on software maintenance), a number of differences are present. The main difference is that we specifically focus on source-code comprehension tasks supported by UML class and sequence diagrams, while the authors consider 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. In some sense, our work fills a gap in that work, explicitly considering those diagrams that are ignored there. Furthermore, the authors do not focus their study on the models produced in a given phase of the development process: models produced in the requirement engineering process and design phase have been considered together. Another difference with respect to our study is that the effect of experience and ability is not analyzed at all.

Dzidek et al. [12] investigate the costs and benefits in using the UML to maintain and evolve software systems. The authors conduct a controlled
experiment with professional programmers. The results reveal that the use of the UML significantly impacts the functional correctness of the maintenance operations. Conversely, the use of the UML does not significantly affect the time to perform maintenance operations. This result is corroborated in our study: the effect of UML diagrams is not significant in task completion time. Then, the use of the UML is not a cause of distraction in case software maintainers have experience with that notation. The main difference with respect to our investigation is that the focus of the controlled experiment is not the comprehension of source-code.

Staron et al. [13] show the results of a series of controlled experiments with students and professionals on UML stereotypes represented by ad hoc icons. The authors assess the effectiveness of these stereotypes in UML class diagrams on the tasks of comprehending object-oriented applications in the telecommunication domain. The use of stereotypes significantly improves the comprehension of the considered applications. As opposed to the present study, the effect of UML behavioral diagrams (i.e., sequence diagrams) is not investigated.

Genero et al. [14] present a controlled experiment with 77 undergraduate students which studies the influence of stereotypes in the comprehension of UML sequence diagrams. The effect of stereotypes is not statistically significant. However, the results show a slight tendency in favor of stereotypes. The effect of sequence diagrams both using and not using stereotypes is not analyzed with respect to source-code comprehension.

2.2. Effect of the ability and experience on UML comprehension tasks

As far as the influence of participants’ ability and experience on the execution of comprehension tasks is concerned, a few empirical investigations have been conducted. For example, Briand et al. [7] establish that training is required to achieve better results when the UML is coupled with the OCL (Object Constraint Language). The authors focus on models produced in the requirements engineering process and consider in their investigation three typical activities: (i) understanding the analysis document; (ii) modifying the analysis document; and (iii) detecting defects in the analysis document. The authors find that the OCL has the potential to improve an engineer’s ability to understand, inspect, and modify a system modeled with the UML. The results also show a significant interaction between participants’ ability and the use of OCL.
Ricca et al. [15] present the results of a series of experiments to assess the effectiveness of the UML stereotypes proposed by Conallen [16]. The data analysis shows that it is not possible to conclude that the use of stereotypes significantly improves the participants’ comprehension on the models. The results also indicate that the participants’s ability significantly interacts with the use of Conallen’s stereotypes. The participants with low ability achieve significant benefits from the use of stereotypes, while participants with high ability obtain a comparable comprehension level with or without stereotypes. Therefore, the authors conclude that stereotypes reduce the gap between low and high ability participants.

Abrahão et al. [6] present the results of a family of five experiments conducted with students and professionals to investigate whether the comprehension of functional requirements is influenced by the use of the UML sequence diagrams exploited to abstract the behavior of use cases. The results show that sequence diagrams significantly improve the comprehension of software requirements in the case of high ability and more experienced participants. Therefore, sequence diagrams help to better comprehend functional diagrams provided that the stakeholders have an adequate experience (with this kind of UML diagrams) and ability. In a different experimental context, the results of that paper are confirmed in the investigation presented here: to benefit from the UML a given experience and ability are needed. The results themselves are perhaps not overly surprising, but this is acceptable, as empirical evidence needs to be reaffirmed and abstracted through several empirical investigations. In the same sense, there is a research contribution to software engineering in this paper.

3. Controlled Experiments

We have conducted a survey on the role of the UML in the Italian software industry [17]. The results suggest that the core business of the interviewed companies mostly concerns the development and the maintenance of software systems implemented with object-oriented programming languages. The greater part of these companies use UML class and sequence diagrams produced in both the requirements engineering process and design phase (referred to in what follows as requirements and design models, respectively). Another result of this survey is that maintenance operations are performed by practitioners with a few years of experience. The companies generally employ people with a Bachelor or a Master degree in Computer Science with
less than 5 years of experience. To perform maintenance operations the companies spend from 1 to 5 person-hours for typical corrective changes\footnote{A reactive modification performed after the delivery of a software system to correct discovered problems.} while the average effort ranges from 10 to 50 person-hours for perfective changes\footnote{A modification performed after the delivery of a software system to improve its performance or maintainability.}.

On the basis of the results of this survey, we started a long-term investigation to understand the contribution of UML class and sequence diagrams in source-code comprehension \cite{10, 18, 19}. This investigation followed two main directions:

- The first direction regards the maintenance and the comprehension of object-oriented software systems when source-code is considered together with UML diagrams produced in the requirements engineering process (i.e., requirements elicitation and analysis). To this end, we conducted a controlled experiment with Bachelor students in Computer Science at the University of Basilicata \cite{18}. The results show that the use of these models does not significantly improve the comprehension of source-code with respect to the use of source-code alone. It could be due to the fact that UML diagrams produced in the requirements engineering process abstract the problem domain of a subject system and miss design/implementation details included later in the development process. On this subject, we have conducted (together with other researchers) a family of four controlled experiments \cite{19}, with the goal of strengthening the findings discussed above. The experiments were carried out with students and professionals from Italy and Spain. The number of participants was 86 with different abilities and levels of experience with the UML. The main finding was: UML models support neither the comprehensibility of source-code nor its modifiability.

- As far as the second direction of our long-term investigation is concerned, we have conducted a controlled experiment and a replication both presented in this paper. The main goal of our study was to assess potential benefits deriving from the use of UML class and sequence diagrams (both produced in the design phase) on the comprehension of object-oriented source-code. These experiments are complementary to
those previously presented [18, 19] since they are focussed on source-code comprehension supported by design models. The preliminary results of the original experiment are presented in [10]. The original experiment and its replication were carried out by following the recommendations provided in [20, 21, 22].

In this section, we show the planning and the operation phases of the two experiments presented in this paper. We followed the guidelines suggested by Wohlin et al. [22]. For replication purposes, we made available on the web an experimental package and the raw data.

3.1. Definition

Applying the Goal Question Metric (GQM) paradigm [23], the goal of the experiments presented in the paper can be defined as follows:

**Analyze source-code comprehension**

**for the purpose of** investigating the support provided by UML class and sequence diagrams produced in the design phase

**with respect to** the comprehension achieved by two categories of maintainers and the task completion time

**from the point of view of** the researchers, in the context of Master and Bachelor students in Computer Science, and

**from the point of view of** the software maintainers, in the context of young/novice software engineers and junior programmers.

3.2. Planning

We used two software systems based on the Model-View-Controller (MVC) architectural model and considered UML structural and behavioral diagrams developed during the design phase. Structural diagrams focus on the static objects that the system will manipulate, while behavioral diagrams concern the software system behavior. In this study, the considered structural diagrams are the UML class diagrams [3]. The UML sequence diagrams are those among the behavioral diagrams on which the study is focused on.

The systems used in both the experiments are:

**Music Shop.** It is a system for handling the sales of a music shop;

---

[4]www2.unibas.it/gscanniello/UMLDesignModSourceCode/
Theater Ticket Reservation. It is for managing the reservation of theater tickets.

Both the systems are desktop applications. The documentation of these two systems and their source-code were realized within a course on Advanced Object-Oriented Programming (AOOP). The documentation was realized by the lecturer of that course. It was realistic enough for small-sized development projects of the following kinds: in-house software (the system is developed inside the software company for its own use) or subcontracting (a sub-contractor develops or delivers part of a system to a main contractor) [24]. The systems were developed by students of the course AOOP. The students were grouped in teams of 4 or 5. In these experiments, we used the source-code that the lecturer selected among the software systems developed in the academic years 2004-2005 and 2005-2006. We asked the lecturer to choose the implementations he considered the best. He opted for the implementations, who achieved the best grade. We took this design choice to reduce as much as possible researchers’ bias and internal and external validity threats [25].

The people who realized the documentation and developed Music Shop and Theater Ticket Reservation were not involved in the experimentation.

3.2.1. Context.
We used the following two groups of participants:

Master’s students. They were last year students enrolled to a Master program in Computer Science at the University of Salerno. These students were hosted for external industrial internships at the end of their Bachelor program and/or are or were industry professionals. Therefore, we can consider them not far from junior programmers [26].

Bachelor’s students. They were second year students of the Bachelor program in Computer Science at the University of Basilicata. They can be considered the next generation of young professional software engineers, developers, and maintainers [21 27].

The participants to the experiment conducted with Master students (UniSa, from here on) were all graduate students with basic software engineering knowledge. They had knowledge of requirements engineering, high and low level design of object-oriented software systems based on the UML, software
development, and software maintenance. We asked the students to accomplish the experiment as an optional activity of an Advanced Software Engineering course of the Master program in Computer Science at the University of Salerno. The experiment conducted with Bachelor students (UniBas from here on) was a differentiated replication on UniSa. Each experiment involved 16 students, whose participation was on voluntary basis.

The UniBas participants were asked to accomplish the experiment within a Software Engineering course. They knew the basics of requirements engineering, high and low level design of object-oriented software systems based on the UML, and object-oriented programming. These students are less experienced than those who participated to UniSa.

The participants were not graded on the results they achieved in the experiments. We asked the participants to perform the tasks based on the experimental objects individually and in a professional way to get one point to their final mark of the exams of the courses above. The data have been collected anonymously.

We selected the two used experimental objects within the software systems discussed above keeping in mind a trade-off between complexity and relevance of the functionality chosen (e.g., the reservation of a ticket for Theater Ticket Reservation). We selected the UML diagrams and the corresponding source-code so that a comprehension task on them did not need more than one hour. The two experimental objects were small enough to fit the time constraints of the experiment though realistic for small maintenance operations that a novice software engineer perform within a software company [17]. The use of incomplete documentation and of a subset of the entire software system on which a maintenance operation impacts is quite common in software industry. The documentation could be incomplete for several reasons. Examples are: only a part of the documentation has been realized, the whole documentation could excessively distract maintainers, and not all the documentation is up-to-date [28]. The experimental objects were also selected to be similar enough in terms of size and complexity. For Music Shop, we selected a chunk of 463 LOCs as the experimental object (S1, from here on). The class diagram of the experimental object contained: 6 classes, 27 attributes, and 45 methods. The sequence diagram depicted 1 actor, 5 objects, and 11 messages. For the second system, we selected a chunk of 378 LOCs as the experimental object (S2, from here on). The associated class diagram contained: 5 classes, 22 attributes, and 36 methods, while the sequence diagram had 1 actor, 5 objects, and 9 messages. The effect of the
experimental objects (e.g., the problem domain of the systems) has been analyzed in the empirical investigation presented here (see Section 3.2.4).

3.2.2. Hypotheses Formulation.

We defined and tested the following null hypothesis:

- **Hn0**: The presence of the UML class and sequence diagrams does not significantly improve the comprehension of source-code.

This hypothesis is one-sided because we assumed a positive effect of the diagrams on the comprehension of source-code [10]. In case the null hypothesis can be rejected, the following alternative one can be accepted:

- **Ha0**: The presence of the UML class and sequence diagrams significantly improves the comprehension of source-code.

In both our experiments, we also defined and tested the following null hypothesis:

- **Hn1**: The presence of the UML class and sequence diagrams does not significantly affect the time needed to accomplish a comprehension task on source-code.

This hypothesis is two-sided because we can make any postulation on the effect of the UML class and sequence diagrams on source-code comprehension. The alternative hypothesis of Hn1 (i.e., Ha1 in the following) can be easily derived on the basis of Hn0 and Ha0.

3.2.3. Variables.

The Control Group denotes the participants who performed the comprehension task with source-code alone. The Treatment Group denotes the participants who performed the comprehension task with source-code and UML diagrams together. Thus, the independent variable (also named main factor or manipulated factor) is Method. It is a nominal variable that can assume two values: NO, Mo (source-code alone) and Mo (source-code and UML diagrams).

Although we expected that the source-code comprehension would increase when the participants are provided with UML diagrams, participants’ ability and experience may affect that comprehension. Therefore, we are also interested in investigating the effect of the following factors and their possible interaction with Method:
Ability. The participants may have different levels of understanding regarding the UML and the modeling of object-oriented systems. Therefore, the students were classified in High and Low, according to the average grades\(^5\) attained in their academic degrees. As suggested in [6] and [15], students with an average below 24/30 are considered to be Low ability participants, otherwise High.

Experience. It indicates the experience the participants had on systems design, UML modeling, and software development. The students were classified in High and Low. The participants to UniSa were classified as High, while those to UniBas as Low. This classification is because the participants to UniSa received more intensive design and development training than those to UniBas. The participants to UniSa studied UML and the modeling approaches in a Software Engineering Course of the Bachelor program first and then in an Advanced Software Engineering course of the Master program. The participants to UniBas only attended an introductory Software Engineering course.

As a consequence of the adopted design (see Section 3.2.4), we had also to consider and analyze the effect of the following co-factors:

System. The systems in which we selected the experimental objects could affect the comprehension in an undesirable way. For example, the problem domain of the systems Music Shop and Theater Ticket Reservation could be confounded with the effect of Method.

Order of Method. The order in which the participants performed the laboratory runs with and without the UML class and sequence diagrams may bias the results. The comprehension of the source-code may improve and the task completion time may diminish when passing from the first task to the next one.

To test the null hypothesis Hn0, we considered a measure based on the comprehension of the source-code. To this end, we asked the participants to fill out a comprehension questionnaire. For each experimental object (i.e., S1 or S2), the comprehension questionnaire is the same independently from the

\(^5\)In Italy, exam grades assume integer values between 18 and 30. The lowest grade is 18, while the highest is 30.
treatment (i.e., Mo and NO_Mo). The questionnaires for S1 and S2 consisted of 15 multiple choice questions, each admitting the same number of possible answers with only one correct. To avoid biasing the results, the questions were formulated in a similar form \[25\]. On the basis of the comprehension questionnaire, we defined and used comprehension as the dependent variable. It was measured as: the number of correct answers provided by the participants on the comprehension questionnaire.

For Hn1, we considered the dependent variable task completion time. It denotes the time (in minutes) a participant spent to complete a comprehension task on source-code. It was recorded directly by the participants noting down start and stop time under the supervision of the authors. This method is widely used in the literature (e.g., \[15\]).

3.2.4. Experiment Design.

For both experiments, we adopted the within-participants counterbalanced experimental design shown in Table \[1\]. We randomly assigned the same number of participants to each group: A, B, C, and D. Each participant worked on S1 and S2 using either Mo or NO_Mo. For example, the participants in the group A started to perform the first comprehension task in the first laboratory run (or trial) on S1 using Mo and then used NO_Mo to perform the second comprehension task on S2 in the second laboratory run. A break of fifteen minutes between the two runs was given.

3.2.5. Execution of the Experiments.

Pilot. Some days before UniSa, a pilot experiment was conducted to evaluate possible issues related to the experimental material. The participants to the pilot were 2 Bachelor students in Computer Science at the University of Basilicata. They did not take part subsequently in UniBas. One of the students performed the comprehension task on S1 using Mo and the second comprehension task on S2 using NO_Mo. The other student accomplished the tasks on S1 and S2 using NO_Mo and Mo, respectively. The results indicated that the experiment was well suited for Bachelor students. We also deduced that two hours suffice for accomplishing the comprehension tasks on S1 and S2 both using NO_Mo and Mo. The participants to the pilot noted minor issues in the experimental material (e.g., some questions of the comprehension questionnaire were not clear). We properly fixed them before conducting UniSa and UniBas.
Experiment Execution. The experiments were organized in three steps. In the first step the participants attended an introductory lesson on how to execute comprehension tasks. Instructions on these tasks and the experimental goals were also presented. The participants were also informed of the pedagogical purpose of the experiments (i.e., using UML models in the execution of source-code comprehension tasks). To avoid biasing the results, the experimental hypotheses were not presented to the participants.

The second and third steps were sequentially performed in the same day. In the second step, we asked the participants to accomplish the two comprehension tasks in the two laboratory runs according to the experimental design shown in Table 1. To perform the comprehension tasks, we provided each participant with the following material:

1. Handouts of the introductory presentation. It included: (i) a set of instructional slides introducing the UML and (ii) examples of models not related with experimental objects;
2. Printout of the tasks. For each object, we provided the participants with the source-code. Depending on the task, we also furnished the UML design models;
3. Printout of the comprehension questionnaires. One for each experimental object;
4. Some sheets of paper and a pencil.

The material in the second run was given to each participant only when he/she had accomplished the task of the first run and when he/she returned back all the experimental material (e.g., comprehension questionnaire and source-code) to the experiment supervisors.

In the third step, we asked the participants to fill out the post-experiment survey questionnaire shown in Table 2. This questionnaire was used to gain enough insight to strengthen and to qualitatively explain the results of the experiments.

3.3. Analysis Procedure

Table 3 summarizes the analyses performed on the data collected. We used the non-parametric Wilcoxon test to reject the null hypotheses Hn0 and Hn1. In case of unpaired analysis, we chose the Mann-Whitney U exact test. We used non parametric tests because of the sample size and (in some cases) the non-normality of the data. These tests are also very robust and sensitive and largely used for studies similar to those presented in this paper [15].
Statistical tests check the presence of significant differences between two distributions, but they do not provide any information about the magnitude of such a difference. We then used the point-biserial correlation $r$ because it is the best way to compute the magnitude of the difference when a non-parametric test is used [29]. In the empirical software engineering fields [30], the magnitude of the effect sizes measured using the point-biserial correlation is classified as follows: small (0 to 0.193), medium (0.193 to 0.456), and large (0.456 to 0.868).

Further, to analyze the probability that a statistical test will reject a null hypothesis when it is actually false, we analyzed the statistical power of the test performed. The statistical power is the probability that a test will reject a null hypothesis when it is actually false. The statistical power is computed as 1 minus the Type II error (i.e., $\beta$-value). The $\beta$-value estimates the probability of erroneously failing to reject a hypothesis that is actually false. A non-parametric alternative is used for the computation of the $\beta$-value. The value 0.80 is considered as the standard for adequacy for the statistical power [31].

We used interaction plots [32] to study the presence of a possible interaction between Ability and Method and between Experience and Method. They are line graphs in which the means of a dependent variable (e.g., comprehension) for each level of one factor (i.e., Method) are plotted over all the levels of the second factor (e.g., Ability). If the lines are nearly parallel, then no interaction is present, and an interaction is present otherwise. Intersecting lines are a clear evidence of an interaction between factors.

To test the effect of Order of Method, we used a method similar to the one suggested by Briand et al. in [7]. Let $\text{Diff(NO}_\text{Mo})$ be the differences for the comprehension values achieved by the participants, who (according to the experimental design) performed the tasks with NO_Mo first and then with Mo, and $\text{Diff(Mo)}$ be the differences for the comprehension values achieved by the participants, who performed the tasks with Mo first and then with NO_Mo. Differently from [7], we applied the non-parametric Mann-Whitney U exact test to verify whether $\text{Diff(NO}_\text{Mo})$ is significantly greater than $\text{Diff(Mo)}$. We expect that $\text{Diff(NO}_\text{Mo})$ is greater than $\text{Diff(Mo)}$ because the participants’ attitude to use the diagrams may improve when passing from the execution of the first comprehension task to the second one. We then tested the null hypothesis $H_0: \text{Diff(NO}_\text{Mo}) < \text{Diff(Mo)}$.

In all the statistical tests, we decided to accept a probability of 5% of committing a Type-I-Error [22]. To investigate the effect of a factor (i.e.,
Ability, Experience, and System) with multiple tests, we applied the Bonferroni correction [33, 32]. For example, when multiple comparisons are used to analyze whether the effect of a factor is statistically significant, the p-values have to be less than $\alpha_{cor} = \frac{0.050}{2} = 0.025$.

We adopted boxplots to show the answers of the post-experiment survey questionnaire graphically. Boxplots are widely employed since they provide a quick visual representation to summarize data [32].

4. Analysis and Results

Some descriptive statistics (i.e., median, mean, and standard deviation) on the comprehension dependent variable are shown in Table 4. These statistics are grouped by experiment and by Method and System. They show that more experienced participants achieved better comprehension values than less experienced participants with Mo as the mean and median values show. In particular, the mean and median values obtained in UniSa are 13.06 and 14, respectively. The mean value is 11.88 and median value is 12 for UniBas. As for NO_Mo, there is not a huge difference between the mean and median values obtained in both UniSa (11.75 and 12, respectively) and UniBas (11.88 and 12, respectively). The descriptive statistics also show that the participants to each experiment achieved nearly the same results on S1 and S2 both with Mo and NO_Mo, respectively. For example, the greater difference is in UniSa on NO_Mo. The difference between the mean values is 0.26, while the difference between the median values is 0.5.

Table 5 reports the same descriptive statistics as in Table 4 for the dependent variable task completion time. These descriptive statistics show that more experienced participants spent less time to accomplish a comprehension task with Mo. The mean values for accomplishing the tasks with Mo and NO_Mo are 30.06 and 23, respectively. As for median, we obtained 27.50 with Mo, while 20 with NO_Mo. This trend also holds for each experimental object. For example, the participants in UniSa spent on average 33.12 minutes with Mo, while 23.38 with NO_Mo. More and less experienced participants spent almost the same time to accomplish a source-code comprehension task with NO_Mo. In particular, the participants in UniSa spent on average 23 minutes, while those in UniBas 21.19. As for Mo, participants in UniBas spent on average more time than those in UniSa (38.69 and 30.06, respectively).
4.1. Influence of Method

Table 6 reports the results of the statistical analysis on the data of both the experiments. The results suggested (second column) that H₀ can be rejected for UniSa (p-value < 0.01). The effect size is large (i.e., 0.62) and the statistical power is 0.91. The participants that benefited from the class and sequence diagrams (Mo > NO_Mo) were 13, while 2 participants obtained worse comprehension values (Mo < NO_Mo). Only one participant achieved the same values for the dependent variable both using Mo and NO_Mo (Mo = NO_Mo). For UniBas, the null hypothesis H₀ cannot be rejected (p-value = 0.57). The number of participants that benefited from Mo were 5 out of 16, while 6 achieved worse comprehension value with Mo. The number of participants that achieved the same comprehension values with Mo and NO_Mo was 5.

For each participant, we also computed the difference between the comprehension value achieved with Mo and that achieved with NO_Mo. This value is positive in case the participant achieved a better comprehension with Mo, otherwise negative. The difference is zero, when a participant achieved the same comprehension values using Mo and NO_Mo. Table 6 also reports some descriptive statistics (i.e., median, mean, and standard deviation) on these differences. The average value of the differences of the comprehension values that the participants achieved when using Mo and NO_Mo is 1.31 for UniSa, while is 0 for UniBas. The medians of these differences are 1 and 0 for UniSa and UniBas, respectively. The minimum and maximum differences are -3 and 4 for UniSa and -3 and 3 for UniBas. The standard deviations of the differences are 1.66 for UniSa and 1.63 for UniBas, respectively.

Table 7 summarizes the result for the dependent variable task completion time. The results of the data analysis showed that H₁ can be rejected for UniBas since the obtained p-value is less than 0.01. The effect size is large (i.e., 0.83) and the statistical power is 1. All the participants to UniBas spent more time to accomplish a source-code comprehension task when provided with the UML diagrams. As far as UniSa is concerned, the effect of Method is not statistically significant on task completion time. However, 10 out of 16 participants spent more time to accomplish the comprehension task with Mo with respect to NO_Mo. In contrast, the other 6 participants spent less time to accomplish the task with Mo.

The average values of the differences for task completion time when using Mo and NO_Mo is about 7 and 18 minutes for UniSa and UniBas, respectively. The median is 9 for UniSa, while it is 16.5 for UniBas. The minimum and
maximum differences are -14 and 50 for UniSa and 4 and 43 for UniBas. The standard deviations of the differences are 16.61 for UniSa and 10.09 for UniBas, respectively. The results of this analysis on the difference highlight with more evidence the fact that less experienced participants were more comfortable with source-code alone with respect to source-code added with UML diagrams. For more experienced participants, the average value of the differences is less than the average value of the less experienced participants, while the standard deviation value is higher. This result suggests that there were participants to UniSa, who were comfortable with UML diagrams, while others not so much. This result gives more strength to the need of analyzing the effect of Ability on both comprehension and task completion time.

4.2. Influence of Ability and Experience

Table 8 shows some descriptive statistics on the comprehension dependent variable grouping the participants by Ability and Experience (i.e., experiment). High ability participants achieved better comprehension values than low ability ones within each experiment and using both Mo and NO_Mo. Furthermore, high and low ability participants to UniSa achieved better comprehension values with Mo than NO_Mo. For UniBas, high ability participants achieved nearly the same comprehension value both using and not using the diagrams. Low ability participants achieved slightly better values with NO_Mo on comprehension. Regarding task completion time (see Table 9), there is not a huge difference between high and low ability participants within each experiment on NO_Mo. The only remarkable difference concerns UniSa and Mo: high ability participants slightly spent more time than low ability participants to accomplish a source-code comprehension task. For high ability participants the median is 30 and the mean is 32.44, while for low ability participants the value for these descriptive statistics are 24 and 27, respectively.

The effect of Ability on the considered dependent variables is not statistically significant within each experiment as the results of the Mann-Whitney test revealed. Then, there is not a statistically significant difference between high and low ability participants for each treatment (i.e., Mo and NO_Mo) and each dependent variable. The p-values range in between 0.03 and 0.51 for comprehension and in between 0.3 and 0.91 for task completion time. Note that we cannot reject the null hypothesis: there is not a statistically significant difference in comprehension between high and low ability participants on NO_Mo. In fact, the p-value is 0.03 and applying the Bonferroni
correction $\alpha_{cor}$ is 0.025.

As for Ability, the plots in Figures 1 and 2 show that the lines are almost parallel (i.e., no interaction is present) in both the experiments, when considering the dependent variable comprehension. Further, high ability participants achieved higher comprehension values regardless of whether or not they used the diagrams to accomplish the comprehension tasks on source-code.

Figure 3 shows an interaction between Ability and Method on task completion time within UniSa. In particular, low ability participants spent on average the same time both using Mo and NO_Mo to accomplish the comprehension task. High ability participants spent more time with Mo than NO_Mo. Regarding UniBas, Figure 4 shows that the lines are almost parallel and High ability participants spent less time than Low ability participants, whatever is the method used.

For Experience, the plots in Figures 5 and 6 indicate an interaction between Method and Experience on comprehension and task completion time, respectively. In particular, the participants to UniSa and UniBas achieved nearly the same comprehension when using NO_Mo. More experienced participants got a source-code comprehension better than that of less experienced participants on Mo. For task completion time, the interaction plot in Figure 5 shows an interaction between Method and Experience. This plot also suggests that less and more experienced participants spent mostly the same time when using NO_Mo. Indeed, more experienced participants slightly spent more time than low experienced ones. For Mo, more experienced participants spent less time with respect to less experienced participants. Summarizing, more experienced participants seem benefit more from Mo than more experience participants.

We can then deduce that the use of the diagrams improved the comprehension of source-code, when participants have an adequate level of experience (i.e., at least a Bachelor’s degree in Computer Science). However, the results of the Mann Whitney test showed that the differences in the comprehension of high and low experienced participants were not statistically significant. For Mo, the p-value was 0.06, while the p-value was 0.68 for NO_Mo. As far as task completion time is concerned, the results suggest that less experienced participants wasted time to read and browse the UML design models without getting an improved comprehension of source-code. For Mo, the p-value is 0.02, so indicating the presence of a statistically significant difference between more and less experienced participants on task
completion time when using the models. The effect size is large (i.e., 0.53), while the statistical power is 0.28. No statistical significant difference was present for NO_Mo (p-value = 0.67).

4.3. Effect of co-factors

We present here the results of the data analysis on the co-factors.

**System** - For UniSa, the Mann Whitney test indicated that there was no significant effect of System on comprehension. In particular, for Mo and NO_Mo the p-values are 1 and 0.871, respectively. Similar results were achieved in UniBas. The p-value is 0.869 for Mo, while it is 0.956 for NO_Mo.

As far as task completion time is concerned, the Mann Whitney test indicated that there was no significant effect of System in UniSa because the p-value is 0.53 for Mo and 0.79 for NO_Mo. Similarly, the Mann Whitney test indicated that there was no significant effect of System when using Mo in UniBas. The p-value is 0.03 (Bonferroni’s correction has been applied). No significant effect of System was also present for NO_Mo (p-value = 0.14).

**Order of Method** - For UniSa and UniBas, the Mann-Whitney test indicated that the order in which the participants performed the comprehension tasks was not statistically significant on comprehension and task completion time. The p-values are 0.12 and 0.39 for UniSa and UniBas, respectively. These results suggest that the participants did not get a significantly better comprehension of the source-code when passing from the first laboratory run to the subsequent one. Similar results were achieved on task completion time. The p-values are 0.96 for UniSa and 0.74 for UniBas, respectively.

4.4. The Results of the Post-experiment Survey Questionnaire

The answers to the post-experiment survey questionnaire of UniSa and UniBas are summarized by means of boxplots in Figures[7] and [8] respectively. Overall, we can observe that the distributions of the answers in both the experiments are similar. In particular, the participants to UniSa and UniBas considered appropriate the time they had to accomplish the tasks in the laboratory trials (the median is 1 for both experiments). They also clearly understood both the objectives and the comprehension tasks they were asked to accomplish: the medians are 1 for UniSa and 2 for UniBas. A neutral judgment on the complexity of S1 and S2 was given (3 is the median for Q4 and Q5 in both the experiments). All the participants found the use of the
UML effective for the comprehension of source-code (2 is the median for Q6 in both the experiments).

4.5. Discussion

The effect of Method was significant for more experienced participants. An average improvement\(^6\) (or benefit) of circa 12% was achieved when the participants accomplished the comprehension task with UML class and sequence diagrams. To accomplish a task with these diagrams, less experienced participants spent on average 44.8% of the time to accomplish the same task with source-code alone.

High ability participants achieved a better comprehension of the source-code with respect to low ability participants independently from their level of experience. The difference between these two groups of participants was not statistically significant. Specifically, more experienced participants achieved a better comprehension of source-code than less experienced participants when the comprehension tasks were performed with the UML diagrams. Without these diagrams, the comprehension level achieved by the participants to UniSa and UniBas on the source-code is close (see Table 4). These results suggest that a certain level of experience is needed to benefit from the use of the UML class and sequence diagrams produced in the design phase when dealing with the comprehension of source-code. As far as task completion time, the results suggested that a given level of experience is needed to avoid participants to be distracted by the diagrams: on Mo less experienced participants spent significantly more time than more experienced participants to accomplish a comprehension task on source-code. Summarizing, less experienced participants did not get an improved comprehension of source-code and wasted time when that code is provided together with UML class and sequence diagrams. On the other hand, more experienced participants got an improved comprehension of source-code and slightly spent more time when source-code is added with these diagrams.

From the descriptive statistics reported in Table 8 and the interaction plots in Figure 2, we can deduce that when participants are less experienced with the UML the ability could make the difference. In particular, we can

\(^6\)Given two values \((a, b)\), the mean percentage improvement of \(a\) is computed as \(\frac{(a-b)}{b} \times 100\). The values \(a\) and \(b\) are the mean comprehension values achieved by the participants on the systems used in the experiments.
note that high ability participants achieved a better comprehension of source-code using UML design models, while low ability did not benefit from these diagrams. A possible motivation for this result might be related to the UML diagrams studied in our investigation and to the kind of systems used in the experiments. This point is subject of future work.

Regarding task completion time, the descriptive statistics (see Table 5) suggest that the participants to UniBas spent on average less time to complete a comprehension task on S1 (i.e., Music Shop) with respect to S2 when using UML class and sequence diagrams. This difference could be due to the presence of possible differences in the familiarity levels of the participants with the application domains of the software systems used. However, the familiarity of less experienced participants with the application domain affected task completion time, but did not affect the comprehension of the source-code when using UML diagrams. Then, it seems that these diagrams reduce the effect of the familiarity with the application domain on source-code comprehension.

4.6. Implications

We adopted a perspective-based approach to judge the practical implications of our investigation. In particular, we based our discussion on the practitioner/consultant (only practitioner in the following) and researcher perspectives [34]:

- The presence of UML design models (i.e., class and sequence diagrams produced in the design phase) yields an average improvement in terms of source-code comprehension of circa 12% in case of more experienced participants. The effect of these models is statistically significant on source-code comprehension. Less experienced participants achieved almost the same source-code comprehension both using or not UML design models. From the practitioner perspective, this result is relevant because it is useless to give additional information to maintainers in case they are not adequately experienced with the UML. From the researcher perspective, it is interesting to investigate what is the experience threshold so that a maintainer can benefit from the use of UML design models.

- For more experienced participants, the presence of UML design models induces no additional time burden. That is, the time to accomplish a
comprehension task is dependent by participants’ experience: the less the experience of participants, the more the time to accomplish a comprehension task with UML diagram is. This result is relevant from both the practitioner and researcher perspectives. For the practitioner, this finding is useful because in case a maintainer has a given UML modeling experience the comprehension of source-code improves when design models are provided, without affecting the completion time. Therefore, design models can be considered a viable means to support the execution of small maintenance operations on a part of the entire software system. This result does not hold for less experienced participants, who spend more time and do not get an improved comprehension of source-code. These results are relevant from the researcher perspective because it would be interesting to investigate how the time spent on UML diagrams is saved to comprehend source-code and vice versa.

- UML diagrams are considered relevant for comprehending source-code independently from the experience of the participants [17, 35]. As we observed in our investigation, the effect of using UML on source-code comprehension is different in case of less and more experienced participants. In addition, UniBas provides an insight into the difference between the perceived usefulness of design models and the effective advantage when using them. This point can be considered relevant for the researcher.

- In case of more experienced participants, 56% of the participants achieved nearly a perfect comprehension of the source-code when using UML diagrams. The comprehension values these participants achieved were 14 or 15 (the highest possible). This result suggests that the possibility of source-code misunderstanding decreases when it is furnished together with UML design models and maintainers are adequately trained on class and sequence diagrams. It is a relevant result for the practitioner.

- The study is focussed on desktop applications for handling the sales of a music shop and for the management of the ticket reservations of a theater. The documentation of these systems were realistic enough for small-sized in-house software and subcontracting development projects. From the researcher perspective, the effect of UML diagrams on different type of systems (e.g., Web applications) represents a possible future direction. This point is relevant for the researcher and the practitioner.
• We consider models/diagrams developed within a university course by people with an adequate UML experience. The use of diagrams recovered from the source-code (e.g., exploiting a reverse engineering tool) or developed by people not adequately trained on the UML could lead to different results. This aspect is relevant for the practitioner, interested in understanding the best way for documenting source-code, and for the researcher, interested in investigating how the quality of the models and their levels of details should affect source-code comprehension.

• We are not sure that the achieved results scale to real and larger software projects. However, the results are encouraging to hope for the best. This point is relevant for the researcher, but also for the practitioner. In fact, practitioners could be interested in understanding if a typical size of a project exists to benefit from UML diagrams produced in the design phase. Our investigation poses the basis for future research work in that direction.

• The UML is widely used in software industry [35, 17]. The achieved results are then useful for all the companies that exploit the UML in the execution of maintenance operations.

• The results presented in this paper and those previously reported [19] show that: UML class and sequence diagrams are useful in source-code comprehension only when they provide design/implementation details given that maintainers have an adequate experience with the UML. These types of diagrams abstract the problem domain of a subject system when used in the requirements engineering process, while they are concerned with the solution domain when exploited in the design phase [36]. This finding is of interest for the practitioner and the researcher.

5. Threats to Validity

The threats that could affect the validity of the results are presented here according to the schema proposed in [22].

5.1. Internal Validity

Internal validity concerns the degree to which conclusions can be drawn about the causal effect of the independent variable/s on the dependent variable/s considered in the investigation.
Interaction with selection. This threat has been mitigated because each group of participants worked on different experimental objects with either Mo or NO_Mo. Further, the participants within each experiment had similar experience with the UML, software system modeling, and computer programming. Additionally, both the kinds of participants found clear the experimental material.

Maturation. Participants might have learned how to improve source-code comprehension and how to reduce the task completion time when passing from the first laboratory run to the subsequent one. The data analysis showed that the order in which the participants performed these two tasks did not significantly affect the comprehension on the source-code the participants achieved and the time to accomplish these tasks.

Diffusion or imitation of treatments. This threat concerns the information exchanged among the participants, while performing each comprehension task and when passing from the first run to the second one. We prevented this in several ways. The participants were monitored by the experiment supervisors, who did not allow the participants to communicate each other. Another issue could be related to the communication among participants in different experiments. The participants to UniSa did not have any opportunity to give information to those in UniBas because they resided in different regions. Further, the participants to UniSa were asked to give back all the experiment material at the end of the experiment.

5.2. External Validity

The main issue of the external validity refers to the possibility of generalizing the results.

Interaction of selection and treatment. The use of students may affect external validity [26, 37, 38, 39]. Threats are related to the representativeness of the participants as compared with professionals. However, the participants’ familiarity with the UML, the application domains of the experimental objects, and the results of the industrial survey presented in [17] suggest that the participants are not far from novice software maintainers and junior programmers. The participants to UniSa were probably better trained in UML modeling than many senior software professionals of small medium software companies in Italy. However, an increasing number of graduates with such modeling skill is being integrated into the software industry and should therefore increase UML capability.
Interaction of setting and treatment. In our case, it concerns with the software systems\(^7\) on which the participants were asked to perform the experimental tasks. The authors were not involved in the realization of the documentation and in the implementation of the system used in the two experiments. Also, the size and complexity of the used experimental objects may affect the validity of the obtained results. The rationale for selecting the used experimental objects relies on the need of simulating actual comprehension tasks related to small maintenance operations that novice software engineers and/or junior programmers may perform in a software company. Larger and more complex experimental objects could excessively overload the participants, thus biasing the results. Nevertheless, it could be also possible that with more complex and larger objects, the help of UML diagrams may be more effective. To analyze this issue, different users’ studies in terms of case studies with professionals are needed. The use of the source-code printout could have negatively affected the comprehension achieved by the participants on the code independently from the method the participants used (i.e., Mo and NO-Mo).

5.3. Construct Validity

Construct validity concerns generalizing the results to the concepts behind the experiment. Some threats are related to the design of the experiments and to social factors.

Interaction of different treatments. The adopted design partially mitigated these threats.

Mono-method bias. We adopted a well known and widely used measure to quantify source-code comprehension.

Confounding constructs and level of construct. More levels than High and Low could be used in the classification of participants’ ability. We are also aware that the use of a different approach to assess participants’ ability could lead to different results.

Evaluation apprehension. We mitigated this threat avoiding to evaluate the participants on their results. The participants were not aware of the

\(^7\)The used software systems (and their documentation) have never undergone maintenance operations. Therefore, the software entropy can be considered low within these systems and within their source-code, in particular. The low level of entropy may positively affect code comprehension [4]. Software entropy is a concern that has never been studied. Thus, it may represent a direction for our future investigations.
Experimental hypotheses.

*Experimenters’ expectations.* We mitigated this threat formulating the questions of the comprehension questionnaires so conditioning their answers in favor of neither Mo nor NO,Mo. All the questions were formulated in a similar way. The post-experiment survey questionnaire was designed using standard approaches and scales [40].

5.4. Conclusion Validity

Conclusion validity concerns issues that may affect the ability of drawing a correct conclusion.

*Reliability of measures.* The used measure allowed us to assess in an objective and repeatable way the comprehension achieved by the participants on source-code used within the two experimental objects.

*Random heterogeneity of participants.* Regarding the selection of the population, we drew fair samples and conducted our experiments with participants belonging to these samples. Another threat related to random heterogeneity of participants could be the number of observations. For example, the number of participants may affect the statistical power of the performed tests. It is worth mentioning that the population allows for $\beta$-values of less than 9% (when rejecting Hn0) and of more than 95% (when Hn0 is not rejected). It is very good in factorial experiment designs.

*Fishing and the error rate.* For UniSa, the null hypothesis Hn0 has been rejected with a p-value less than 0.01 and 0.91 as the statistical power value. Similarly, Hn1 has been rejected with a p-value less than 0.01 in UniBas. In the data analysis, we used the Bonferroni correction when needed.

6. Conclusion and Future Work

In this paper, we have presented the results of a controlled experiment and of a differentiated replication both conducted to assess whether the comprehension of source-code increases/decreases when UML class and sequence diagrams are present. The diagrams can be considered those produced in the design phase of a UML based-development process [41, 42]. Therefore, our results should not hold for self-managing and autonomic systems (e.g., [43]).

The goal of our long-term investigation (i.e., the family of experiments presented in [19] and the experiments reported here) is significant for software industry. Software managers and engineers have to be convinced that
UML-based modeling is really worth the effort and under which conditions it produces more benefits [11].

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 comprehension 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 the years (e.g., [11, 44]).

The data analysis revealed that more experienced participants (i.e., Master students in Computer Science) benefited from the use of UML design models (class and sequence diagrams). This arises an interesting research issue: in the case of more experienced participants is it better to use class and sequence diagrams together or each of them alone? We preliminarily study this point in [45] and the achieved results suggested that: it is better to use class and sequence diagrams together.

The results presented in this paper also indicated that high ability participants achieved on average better comprehension of source-code than low ability participants, but this difference is not statistically significant. The results support those of similar experiments (e.g., [7, 15]). Another relevant result is that less experienced maintainers waste time to read UML documentation without a significant improvement of source-code comprehension.

The results presented in this paper also suggest the following future research directions:

- **Readability of the UML diagrams.** Less experienced participants could get sidetracked by the large number of details in the class and sequence diagrams. This represents another issue for the external validity that should be controlled in future empirical investigations. We plan to base our future work on the outcomes presented in [46, 47].

- **Participants’ countries and cultures.** It will be interesting to investigate whether the comprehension of source-code supported by UML models is affected or not by the kind of participants. In particular, it will be worth interesting to study whether participants from different countries and having different cultures achieve different comprehension levels on source-code.
• *Industry Professionals*. Replications with professional programmers are needed to confirm or contradict the results of our empirical study. It would be interesting to study also the effect of professionals accustomed to using a given development process with respect to another (e.g., RUP [42] vs. XP [48]).

**Acknowledgments**

We wish to thank the Ilaria Bilancia and Michela Continanza for their precious help in conducting the experiments. We also would like to thank all the participants in the experiments.

**References**


URL http://doi.acm.org/10.1145/2629457


URL http://books.google.at/books?id=72BsZFZmosoC


Figure 1: Analysis of Ability on Comprehension for UniSa, using the interaction plot

Figure 2: Analysis of Ability on Comprehension for UniBas, using the interaction plot

Figure 3: Analysis of Ability on Task Completion Time for UniSa, using the interaction plot

Figure 4: Analysis of Ability on Task Completion Time for UniBas, using the interaction plot

Figure 5: Analysis of Experience on Comprehension, using the interaction plot

Figure 6: Analysis of Experience on Task Completion Time, using the interaction plot

Figure 7: Boxplots of the answers to the post-experiment survey questionnaire for UniSa

Figure 8: Boxplots of the answers to the post-experiment survey questionnaire for UniBas
| Table 1: Experiment design |
| Table 2: Post-experiment Survey Questionnaire |
| Table 3: Performed Analyses |
| Table 4: Descriptive Statistics of the Comprehension Dependent Variable |
| Table 5: Descriptive Statistics of the Task Completion Time Dependent Variable |
| Table 6: Analysis on the Comprehension Dependent Variable |
| Table 7: Analysis on the Task Completion Time Dependent Variable |
| Table 8: Descriptive Statistics on the Comprehension Dependent Variable Grouping Participants by Ability and Experiment/Experience |
| Table 9: Descriptive Statistics on the Task Completion Time Dependent Variable Grouping Participants by Ability and Experiment/Experience |