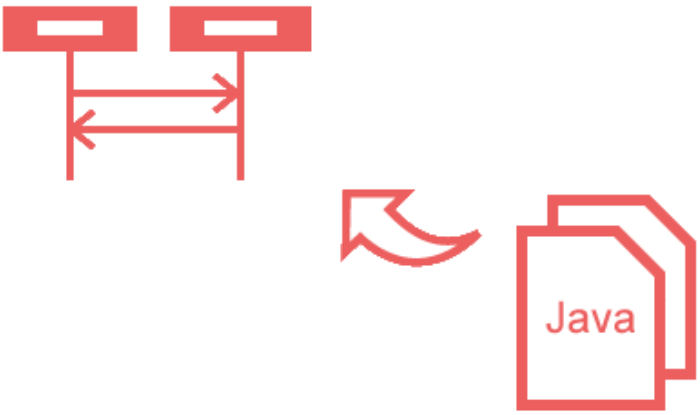
Studying the benefits of using UML on software maintenance : an evidence-based approach

Fernandez Saez, A.M.

CHAPTER 4. ARE FORWARD DESIGNED OR REVERSE-ENGINEERED UML DIAGRAMS MORE HELPFUL FOR CODE MAINTENANCE?



Fernández-Sáez, A. M., Genero, M., Chaudron, M. R. V, Caivano, D., Ramos, I. (2015). Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?: A Family of Experiments. *Information and Software Technology*. 57(1) pp. 644–663, (**JCR Index 2015 = 1.569**).

ABSTRACT

Context: Although various success stories of model-based approaches are reported in literature, there is still a significant resistance to model-based development in many software organizations because the UML is perceived to be expensive and not necessarily cost-effective. It is also important to gather empirical evidence in which context and under which conditions the UML makes or does not make a practical difference. **Objective:** Our objective is to provide empirical evidence as to which UML diagrams are more helpful during software maintenance: Forward Designed (FD) UML diagrams or Reverse Engineered (RE) UML diagrams. **Method:** We carried out a family of experiments which consisted of one experiment and two replications with a total of 169 Computer Science undergraduate students. **Results**: The individual data analysis and the meta-analysis conducted on the whole family, show a tendency in favor of FD diagrams and are significantly different as regards the effectiveness and efficiency of the subjects who participated and played the role of maintainers. The analysis of the qualitative data, collected using a post-experiment survey, reveals that the subjects did not consider RE diagrams helpful. **Conclusions**: Our findings show that there are some objective results (descriptive statistics or statistical tests) related to the maintenance effectiveness and efficiency in favor of the use of FD UML diagrams during software maintenance. Subjective opinions also lead us to recommend the use of UML diagrams (especially class diagrams) created during the design phase for software maintenance because they improve the understanding of the system in comparison with RE diagrams. Nevertheless, we can only assume that these results are valid in the context of Computer Science undergraduate students when working with small systems related to well-known domains, and other contexts should be explored in order to reaffirm the results in an industrial context by carrying out replications with professionals.

Keywords: Software Maintenance; UML Diagrams; Reverse Engineering; Maintainability; Family of Experiments; Controlled Experiment; Survey.

4.1. INTRODUCTION

The benefits of using software documentation to comprehend and modify source code have been widely studied (Abbes et al., 2011; de Souza et al., 2005; Tilley and Huang, 2003). The authors of (Tryggeseth, 1997) report that having documentation available during system maintenance reduces the time needed to understand how to perform maintenance tasks by approximately 20 percent.

UML (OMG, 2010) has become the de facto standard modeling notation used to complement software documentation as a graphical notation. It first appeared in 1997 and has now become one of the most widely-used modeling languages in industry. There is empirical evidence of its benefits during software development because it increases the understanding between customer and developer and improves communication among team members (Nugroho and Chaudron, 2009). UML also improves the source code quality by reducing its defect density (Nugroho et al., 2008).

There is also some evidence of the UML's benefits during software maintenance. The availability of UML documentation may result in significant improvements to the functional correctness of changes in addition to their design quality. However, it does not appear that any time is saved as a result. For simpler tasks, the time needed to update the UML documentation may be substantial in comparison with the potential benefits, thus motivating the need for UML tools with better support for software maintenance (Arisholm et al., 2006; Dzidek et al., 2008).

Despite these benefits, there are some common modeling problems that lead to models being less effective and less adopted (Berenbach and Konrad, 2007). For example, some of the limitations of program understating and maintenance supported by UML are the following:

- unclear specifications of syntax and semantics in some of the UML's more advanced features (Tilley and Huang, 2003), leading to the need for adequate training in UML (Anda et al., 2006),
- spatial layout problems, e.g., large diagrams are not easy to read (Tilley and Huang, 2003),
- the UML's insufficient support as regards representing the domain knowledge required to understand a program (Tilley and Huang, 2003),
- The time spent updating the UML diagrams according to the changes in source code counteracts the improvement in source code maintenance time (Arisholm et al., 2006; Dzidek et al., 2008).
- UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability (Scanniello et al., 2014).

Despite its limitations, there are clear benefits of using UML, such as the improvement in the traceability from functional requirements to code (Anda et al., 2006), and the fact that UML is always beneficial in terms of functional correctness (reducing defect density) (Nugroho and Chaudron, 2007).

Moreover, diagrams with a high level of detail are reported to be more helpful during software development (Nugroho, 2009), while those with a low level of detail seems to be better when performing maintenance tasks (Fernández-Sáez et al., 2012).

As we can see, there is some evidence to support the use of UML modeling during software maintenance, but the results are not conclusive owing to the diversity of the results obtained in different empirical studies. It is not clear whether an investment in creating and updating UML diagrams has a return during later phases, like software maintenance which consumes a high percentage of a project's resources. If the presence of UML diagrams is a clear positive factor, but it is not clear what the best way to create them is, then why not create them directly from source code and save the time needed for their creation and update? This is related to a possible influential factor in the software maintenance which has not being studied before: the Origin of the diagrams. FD diagrams, i.e., the diagrams generated during forward development, are sometimes available for maintainers in the maintenance phase, but when this is not the case, the diagrams may be reconstructed through an RE technique (Perez-Castillo

et al., 2011). The difference in the origin of the diagrams (i.e., FD diagrams or RE diagrams) and the different techniques that can be used to generate an RE diagram result in different styles of diagrams that may influence the quality of the source code being maintained.

RE diagrams are easier to obtain than FD diagrams because they can be automatically generated with a reversing tool without high investments in developer effort (although these kinds of diagrams need some polishing). Given the ease of their generation and that they can be generated automatically at any time, maintainers can have up-to-date diagrams that model the system when they need them. However, the problem with these diagrams is their very high level of detail, which may lead to difficulties in understanding them. There are several issues related to the obtaining of diagrams with a high level of detail when they originate from source code, after applying a reverse engineering technique:

- The level of abstraction is very low, owing to the fact that every element in the source code is represented in the UML diagrams. The benefit of this is that there is a very high traceability from the diagrams to the source code.
- The business rules allow the designers to create UML diagrams by following a specific design objective. The developers then implement the source code by following these diagrams. RE diagrams do not represent these rules, since that they are obtained from source code and these diagrams only reflect *how* the code was implemented, rather than *why*.
- These RE diagrams, unlike FD diagrams, are platform-dependent and therefore contain details about the implementation patterns and frameworks used, which do not appear in FD diagrams.
- After obtaining the RE diagrams, a cleaning and lay-outing process needs to be performed in order to adapt them to their corresponding audience.

However, there is another option when up-to-date diagrams are required: the maintainer may keep the source code and the diagrams in-synch manually by applying the corresponding changes incurred by maintenance to both. This option requires more manual effort than the RE process, because the process is not as automated as the RE approach is. Nevertheless, when the diagrams are generated by people and not by automated tools, they may contain different levels of abstraction and detail, depending on the importance of diagram elements (Osman et al., 2012); this may make diagrams more understandable, and hence more effective.

All of the above lead us to pose our main research question: "Should software maintenance companies spend time updating their UML diagrams or should they use Reverse Engineered diagrams instead?" Our results might be useful for companies that are performing software maintenance and yet are unsure whether they should continue updating their UML diagrams (as part of the project documentation) or whether they might save that time by automatically generating RE diagrams.

On the one hand, if we obtain better results with design UML diagrams we will have empirical evidence to encourage companies and software developers to follow a

model-centric approach. This implies beginning the development of a software system by building the corresponding UML diagrams and keeping them up-to-date, thereby facilitating maintenance tasks. On the other hand, if we obtain better results with RE diagrams, we will have empirical evidence to suggest that maintainers should obtain the UML diagrams needed by using RE techniques. This will thus avoid the need to maintain the available diagrams (whenever these are available) and will reduce the time involved in maintenance tasks.

There is a need for experiments focused on software maintenance and evolution by a non-original developer, as this consumes the majority of the resources in a typical software organization.

All of the above led us to perform a family of experiments to investigate whether the different *Origins* of UML diagrams (Reverse Engineered or Forward Design diagrams) affect the maintainer's performance when modifying source code. Our aim is to discover whether or not, in order to obtain an up-to-date version of diagrams, an effort should be made to maintain diagrams. The family consists of a controlled experiment, which was previously presented in (Fernández-Sáez et al., 2013b), and two replications, carried out with students from two countries (Spain, and Italy). The participants were 169 Computer Science undergraduate students with different abilities and levels of experience with the UML and with JAVA source code.

The three main goals of this paper are therefore to present a thorough description of this family of experiments, its main findings and their practical implications.

This paper is organized as follows. Section 4.2 presents the related work. Section 4.3 provides a description of the experiment and replications. The results obtained in the experiment are set out in Section 4.4, whilst the implications of the study are summarized in Section 4.5 and the threats to validity are summarized in Section 4.6. Finally, Section 4.7 outlines our main conclusions and future work.

4.2. RELATED WORK

The related work will be focused on the empirical evidence on the use of UML diagrams in software maintenance. A Systematic Literature Review that was carried out to collect all the empirical studies performed as regards the use of UML in maintenance and the understandability of UML diagrams which may influence the maintenance of the system is presented in (Fernández-Sáez et al., 2013a). In this SLR, 46 primary studies were found reporting 74 empirical studies related to this topic, but only the following works directly related to the use of UML diagrams in source code maintenance:

- In (Dzidek et al., 2008) an experiment was performed to investigate whether the use of UML influences maintenance in comparison to the use of only source code. This experiment investigates the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real nontrivial system, using 20 professional developers as subjects. These maintainers had to perform 5 maintenance tasks consisting of adding new functionalities to the

existing system after which the correctness, time and quality of the solution were measured. Both source code and UML diagrams, when available, had to be maintained. The results of this work show a positive influence of the presence of UML for maintainers. In terms of time, the UML subjects took more time if the UML documentation had to be updated but that difference was not statistically significant. However, UML was always beneficial in terms of functional correctness (introducing fewer faults into the software) because the subjects in the UML group had, on average, a practically and statistically significant 54 percent increase in the functional correctness of changes. UML also helped produce code of a better quality when the developers were not yet familiar with the system. This experiment is a replication of a previous work performed with students which is presented in (Arisholm et al., 2006) and which obtained similar results. The main difference between (Dzidek et al., 2008) and the work presented here is that (Dzidek et al., 2008) shows how source code is maintained when UML diagrams are complementing the source code or when the source code is alone, while the subjects of the family of experiments presented herein always received UML diagrams (some received FD diagrams while others received RE diagrams). Another difference is that the UML diagrams used in our family of experiments did not need to be updated according to source code changes. In addition, the subjects of (Dzidek et al., 2008) were professionals.

- Arisholm et al. (Arisholm et al., 2006) presented the results of two controlled experiments carried out to assess the impact of UML design diagrams on software maintenance. 98 undergraduate students 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 diagrams. As in (Dzidek et al., 2008), the participants' ability and experience were not analyzed with regard to the comprehensibility and modifiability of source code. Unlike our study, 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 on both UML diagrams and source code. The differences with the work presented in (Arisholm et al., 2006) and our work are similar to those mentioned in the case of (Dzidek et al., 2008), except for the type of subjects involved.

It is important to mention some other important papers relating to the influence of the use of UML diagrams during software maintenance which were not found as part of (Fernández-Sáez et al., 2013a) because of their dates of publication or because of the objective of the SLR:

- Scanniello et al. (Scanniello et al., 2014) used a family of controlled experiments to discover that the use of analysis UML diagrams (those obtained in an early phase

or the development process, such as the requirements elicitation or analysis phase) does not significantly improve the comprehension and modifiability of source code with regard to the use of source code alone. These results are valid in the context of undergraduate students and small size systems related to well-known domains. The study presented herein is, however, focused on FD and RE diagrams, which have a higher level of detail. Moreover, the dependent variables are different because the work of Scanniello et al. focused on the comprehension and maintainability of the source code, while the work presented here is focused solely on the maintainability of source code, and bigger systems are also considered.

- In (Scanniello et al., 2012) the results of an experiment to assess whether the comprehension of source code is affected when it is added to UML class and sequence diagrams produced in the design phase is presented. The results reveal that the participants benefited from the use of the UML diagrams. An average improvement of 14% was achieved when the participants accomplished the comprehension task with the class and sequence diagrams, but the time needed to comprehend the code was not significantly influenced from a statistical point of view. Our work differs from that presented in (Scanniello et al., 2012) because we compare FD diagrams with RE diagrams rather than with no UML diagrams, and the dependent variable is also different because (Scanniello et al., 2012) focuses on the comprehension of the source code and the work presented here is focused on the maintainability of source code. Bigger systems are also used here.
- In the work presented in (Karahasanovic and Thomas, 2007), the experiment performed is focused on the comprehension and the difficulties involved in maintaining object-oriented systems. During this experiment the subjects, who were 34 students in their third year of a computer science degree, had to perform 3 different maintenance tasks on a medium-size object-oriented system written in Java. The tasks were related to extending, updating and deleting functionalities of the system, i.e., maintaining the system. It is important to note that the subjects had to give higher priority to the quality of the solutions than to a shorter development time, which may have influenced the results of the experiment. UML diagrams were also presented to the subjects of the experiment, and the correctness of their solutions was measured, but this work was only focused on exploring the participants' strategies and problems while they were conducting maintenance tasks on an object-oriented application. Two major groups of difficulties were related to the comprehension of the application structure, namely the understanding of GUI implementation and OO comprehension and programming. The main difference between the work presented in (Karahasanovic and Thomas, 2007) and the work here is that the subjects' performance was not measured in the former while it is measured in the latter. In (Karahasanovic and Thomas, 2007) the results are qualitative (description of problems while maintaining source code), while here they are quantitative (results on performance, such as time spent and correctness of the answer).

- The authors of (Scanniello et al., 2010) show the results of an explorative survey used to investigate the state of the practice regarding the use of UML in software development and maintenance. The majority of the companies interviewed use UML for software development and to perform maintenance operations. Maintenance operations are mainly performed by practitioners with little experience. Another interesting point concerns the average effort needed to perform maintenance operations, which ranges from 1 to 5 person hours for an ordinary maintenance operation (e.g., corrective changes), and from 10 to 50 person hours in the case of an extraordinary maintenance operation (e.g., perfective or adaptive changes). The differences between the work presented in (Scanniello et al., 2010) and ours are based on the nature of the empirical study (an explorative survey vs. an explicative family of experiments).
- An experiment presented in (Fernández-Sáez et al., 2012) studies whether different Levels of Detail (LoD) in UML diagrams might influence the maintenance of source code. In (Fernández-Sáez et al., 2012; Nugroho, 2009) there is an assumption that the higher the amount of information put into a diagram, the more is known about the concepts/knowledge described in it. That being the case, a higher LoD would improve maintainers' performances owing to the fact that they would understand the system they have to maintain better. The results from (Fernández-Sáez et al., 2012) are not conclusive, but show a slight tendency in favor of high LoD diagrams. The similarities of the work presented in this paper with (Fernández-Sáez et al., 2012) arise from the fact that FD diagrams could be considered as having a high LoD, while RE diagrams could be considered as having a very high LoD. The difference between (Fernández-Sáez et al., 2012) and the work presented here are the independent variables used (LoD of UML diagrams in the former, and the origin of UML diagrams in the latter), and the design of the experiment (within-subjects and between-subjects respectively).

Having assumed that the presence of UML diagrams is a positive factor for software maintenance, it is important to know what kinds of UML diagrams are better at improving these kinds of tasks. It is therefore important to additionally study the comprehension of UML diagram in themselves. It is possible to find many papers related to the comprehension of UML diagrams (which is directly related to the comprehension of the software system) in literature (Dobing and Parsons, 2006; Genero et al., 2011). We can highlight two papers found as part of (Fernández-Sáez et al., 2013b), which focus solely on software development using different kinds of UML diagrams during this phase but which might also be related to maintenance:

- The study presented in (Nugroho, 2009) consists of an empirical experiment focused on determining how different Levels of Detail (LoD) influence the understandability of UML diagrams. It measures the correctness and efficiency in comprehending UML diagrams of 53 Computer Science Master's students. The results show a better understanding of diagrams when they have a high LoD. There are some differences between the work presented in (Nugroho, 2009) and the

family of experiments summarized in this paper. On the one hand, the experiment presented in (Nugroho, 2009) focuses solely on the comprehension of UML diagrams, and the subjects were not given the source code of the system. The experiment presented in (Nugroho, 2009) is similar to that presented in (Fernández-Sáez et al., 2012), but focuses on the development of software rather than the maintenance of software and with the difference that (Nugroho, 2009) is focused on the comprehension of the UML diagrams and (Fernández-Sáez et al., 2012) on the maintenance of source code.

- In (Tilley and Huang, 2003) an experiment with 15 subjects (PhD students or Professors) to assess the qualitative efficacy of UML diagrams in aiding program understanding is described. The experiment had participants analyze a series of UML diagrams and answer a detailed questionnaire concerning a hypothetical software system. Results from the experiment suggest that the relation between the correctness of the solution and the time spent obtaining it using UML to support program understanding is limited by factors such as ill-defined syntax and semantics, spatial layout, and domain knowledge.

All related work that is relevant to this paper is summarized in Table 4.1to provide the reader with a better extract of the main information in order to compare the empirical studies. The columns of the table are described as follows:

- **Ref**: contains the reference to the paper that presents the empirical study considered.
- **Type of empirical study**: indicates the type of empirical study summarized in the paper (a survey, an experiment, a family of experiments, etc.)
- **Goal**: describes the goal pursued by the empirical study.
- **Subjects**: presents the numbers of subjects who participated in the empirical studies and the type of subjects (students, professionals, academic staff, etc.).
- **Independent variables**: describes the variables that are studied to ascertain their effect on the dependent variables. The values (treatments) of the independent variables are also presented.
- **Dependent variables**: presents the outcome variables, which are the variables that are affected by the changes produced in the independent variables.
- **Experiment design**: contains the type of design selected, which can be between-subjects (each subject receives only one treatment) or within subjects (each subject receives all the treatments).
- **Tasks**: describes the tasks to be performed by the subjects as part of the empirical study.
- **Results**: reveals the main findings obtained.

Table 4.1. Summary of the related work (part 1/2).

Ref	(Dzidek et al., 2008)	(Arisholm et al., 2006)	(Scanniello et al., 2014)	(Scanniello et al., 2012)
Study	1 experiment	2 experiments	Family of experiments (1+3)	1 experiment
Goal	Investigates whether the use of UML influences maintenance in comparison to use of only source code	Investigates whether the use of UML influences maintenance in comparison to the use of only source code	Investigates whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code.	Investigates whether the comprehension of source code increases when participants are provided with UML class and sequence diagrams produced in the software design phase
Subjects	20 professional developers	Undergraduate students (22 and 76, respectively)	Undergraduate students (24, 22, 22 and 18)	16 undergraduate students
Independent variables	The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code)	The use of UML documentation in a UML-supported IDE(possible values: presence or absence of UML diagrams accompanying source code)	The use of UML analysis models(possible values: presence or absence of UML diagrams accompanying source code)	The use of sequence and class diagrams created in the design phase(possible values: presence or absence of UML diagrams accompanying source code)
Dependent variables	Time spent modifying source code. Time spent modifying source code + UML diagrams. Functional correctness and quality of the solution.	Time needed to change source code. Time needed to change source code + UML diagrams. Correctness of the change. Quality of the change.	Comprehension of level of the source code. Capability of a maintainer to modify source code.	Comprehension of source code.
Design	Between-subjects	Between-subjects	Within- subjects	Within-subjects
Tasks	Modify source code and UML diagrams	Modify source code and UML diagrams	Comprehension and modification tasks. Post-experiment questions	Comprehension questions
Results	The subjects who received UML diagrams needed more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce code of a better quality when the developers were not yet familiar with the system.	The subjects who received UML diagrams needed more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce code of a better quality when the developers were not yet familiar with the system.	UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability.	Participants comprehend source code significantly better when it is added to class and sequence diagrams together.

Table 4.1. Summary of the related work (part 2/2).

Ref	(Karahasanovic and Thomas, 2007)	(Scanniello et al., 2010)	(Fernández-Sáez et al., 2012)	(Nugroho, 2009)	(Tilley and Huang, 2003)
Type of study	1 experiment	1 survey	1 experiment	1 experiment	1 experiment
Goal	Investigates the comprehension and the difficulties involved in maintaining object-oriented systems	Investigates the state of the practice regarding the use of UML in software development and maintenance in Italian industry	Determines the influence of different levels of detail (LoD) of UML diagrams in source code maintenance	Determines the influence of different levels of detail (LoD) on the understandability of UML diagrams	Assesses the qualitative efficacy of UML diagrams in aiding program understanding
Subjects	34 undergraduate students	-	11 undergraduate students	54 Master's students	15 subjects (PhD students or Professors)
Independent variables	-	-	The level of detail of UML diagrams (possible values: high or low LoD)	The level of detail of UML diagrams (possible values: high or low LoD)	-
Dependent variables	Comprehension and modification of systems	-	Understandability and modifiability of source code	Comprehension of UML diagrams	Comprehension of UML diagrams
Design	Between-subjects	-	Between-subjects	Between-subjects	Between-subjects
Tasks	Modification questions	-	Comprehension tasks + modification tasks + subjective questions	Comprehension tasks	Comprehension questions + modification questions
Results	Detection of common difficulties in understanding when maintaining software systems.	The majority of the companies interviewed use UML for software development and to perform maintenance operations. Maintenance operations are mainly performed by practitioners with little experience.	No significant results in favor of high or low LoD. There is a slight tendency in favor of low LoD diagrams	Results show that the effect of LoD in UML models on model comprehension is significant in favor of high LoD	UML's efficacy in support of program understanding is limited by factors such as ill-defined syntax and semantics, spatial layout, and domain knowledge

It is clear that the LoD of UML diagrams is an important factor which has previously been studied in literature because the LoD of UML diagrams influences their understanding (and hence also source code understanding), and adding details to a diagram is a time consuming task. On the one hand, low LoD diagrams, like analysis models, do not appear to be helpful for source code maintenance (Scanniello et al., 2014). On the other hand, high LoD diagrams are better understood than low LoD diagrams when performing software development (Nugroho, 2009). It is therefore clear that a balanced LoD will be the best option, but there is a tendency to use diagrams with a detailed designed, especially when FD are used (Scanniello et al., 2012). But how many details should be presented in a UML diagram? If a high LoD diagram is the best option, why not use an RE diagram, which can be automatically generated (saving time) in comparison to creating UML diagrams during software development and keeping them up-to-date? These questions motivated us to plan a family of experiments that compare the usefulness of FD and RE diagrams when performing source code maintenance.

4.3. DESCRIPTION OF EXPERIMENT

Families of experiments allow researchers to answer questions that are beyond the scope of individual experiments and permit the generalization of findings across studies, thus providing evidence with which to for confirm or reject specific hypotheses (Basili et al., 1999). Replications of empirical studies might be regarded as an essential activity in the construction of knowledge in any empirical science based on the following propositions: "We do not take even our own observations quite seriously, or accept them as scientific observations, until we have repeated and tested them" (Popper, 1959) and "… replication is needed not merely to validate one's findings, but more importantly, to establish the increasing range of radically different conditions under which the findings hold, and the predictable exceptions".

In order to run and report this family of experiments, we followed the recommendations provided in several pieces of work (Jedlitschka et al., 2008; Juristo and Moreno, 2001; Wohlin et al., 2012). The family of experiments followed the guidelines for reporting empirical research in software engineering (Jedlitschka et al., 2008) as closely as possible. The experimental material is available for downloading at: http://alarcos.esi.uclm.es/originUMLmaintenance/

In the following subsections we shall describe the main characteristics of the experiment and the replications, including goal, context, variables, subjects, design, hypotheses, material, tasks, experiment procedure and analysis procedure. Figure 4.1 presents the chronology of the family of experiments, summarizing the name of the experiment, the number of participants, and the name of the Universities at which the experiment and replications were run.

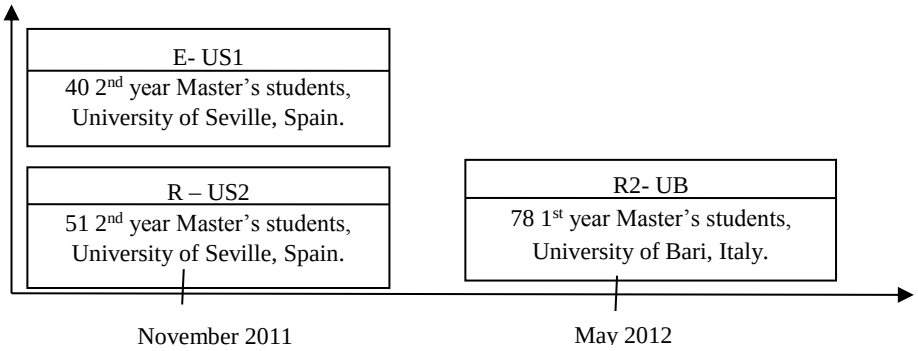


Figure 4.1. Chronology of the family of experiments.

4.3.1. Goal

The principal goal of this family of experiments was to investigate whether the *Origin* of UML diagrams influences the maintenance of source code. The GQM template for goal definition (Basili and Weiss, 1984; Basili et al., 1999) was used to define the goal of our experiment as follows: "*Analyze* the maintainability of source code *from the point of* view of software maintainers *with regard to* the Origin of the UML diagrams, *in the context of* Computer Science students at the Universities of Seville and Bari".

We considered two possible *Origins* of the diagrams: the design phase and an RE technique. In the first case, our intention was to maintain the source code using the UML diagrams built during the design phase. In the second case, we set out to maintain a source code for which the UML diagrams were not available, which meant that they would have to be obtained from the source code using an RE technique.

We decided to consider class diagrams and sequence diagrams because they can be obtained from an RE technique and because they are also two of the most commonly used diagrams when designing a system (Dobing and Parsons, 2006; Erickson and Siau, 2007; Grossman et al., 2005).

4.3.2. Context selection

The experimental objects consisted of class and sequence diagrams and the Java code of one system. The diagrams were obtained from different *Origins*:

- **RE**: Reverse Engineered UML diagrams, which are constructed completely automatically based on the source code.
- **FD**: Forward designed UML diagrams obtained at the design phase. These are totally manually designed.

RE diagrams are diagrams with a high LoD since they represent all the elements in the source code. The FD diagrams might also be considered as diagrams with a high LoD because their class diagrams contain class names, attributes, operations and relationships, and their sequence diagrams contain lifelines, messages and parameters of messages. However, FD diagrams do not represent all the elements in the source

code, but those elements which are represented (based on human selection) are completely represented. FD diagrams can therefore be considered as high LoD diagrams while RE are higher LoD diagrams.

The diagrams described a sports center system from which users can rent services (tennis courts, etc.). The system is a sports center application which was created as part of the Master's degree Thesis of a student from the University of Castilla-La Mancha, and we therefore consider it to be a realistic system. It is a desktop application created with the client-server paradigm. The system contains 5123 Lines of Code (LoC) (Table 4.2), so it might be considered a small realistic system. In fact its size is almost double the LoC of other systems used in previous works which have nevertheless been considered as realistic systems, for example in (Dzidek et al., 2008). The maintenance requirements were formulated by the Master's supervisor. In the case of the FD diagrams, 4 class diagrams are available, with a total of 16 classes, and 21 sequence diagrams, with 226 messages. In the case of RE diagrams, 4 class diagrams are available, with 21 classes, and 11 sequence diagrams, with 191 messages. The number of classes in class diagrams is a good deal smaller than in the class diagrams of systems used in other previous work. This is owing to the use of different levels of abstraction for modeling, but their diagram size is still representative of realistic systems (Heijstek and Chaudron, 2009). Note that the number of sequence diagrams in the RE group is 11 and the number of diagrams in D group is 21. The number of messages per diagram (226 messages for 21 diagrams in D group, and 191 messages for 11 diagrams in RE group) therefore provides us with an indicator which suggests that RE diagrams should be considered as being larger and more complex. The RE diagrams were generated using the IBM Rational Software Architect tool, employing the default RE-functionality that it provides, followed by auto-lay-outing (also offered by the same tool). These experimental objects were presented in Spanish in E-US1 and R-US2. A native Italian speaker translated all the material into Italian in the case of R-UB. The replicators supported the native speakers and helped them when needed (e.g., in the translation of technical terms).

Table 4.2. Metrics of the software system used in the family of experiments.

	#Class diagrams	#classes	#Sequence diagrams	#messages	LoC
D	4	16	21	226	5123
RE	4	21	11	191	

The participants, who were grouped by experiment, had the following characteristics:

- **E-US1**: The participants were 40 Computer Science students from the University of Seville (Spain) who were taking the Software Engineering III course in the second-year of their Master's Degree, from which they had acquired training in UML diagrams (as they also had from previous Software Engineering courses). 90% of the students were non-repeating course students and most of them did not have industrial experience (five subjects had experience as programmers, another as an analyst and another as a maintainer). The subjects considered their

knowledge of the UML to be high/very high in most cases for class diagrams, and medium/high for sequence diagrams (the two kinds of diagrams which were used during the experiment). Most of the subjects' knowledge of JAVA language is excellent (all the subjects considered it to be above average). In Spain, grades are expressed as doubles and assume values of between 5 and 10. The lowest grade is 5, while the highest is 10.

- **R-US2**: The participants were 51 students from the University of Seville (Spain) who were taking the same course as the students in the original experiment, i.e., they were taking the Software Engineering III course in the second-year of their Master's Degree, from which they had acquired training in UML diagrams (as they also had from previous Software Engineering courses).In this case, more than half (58%) the students were repeating the course and most of them had industrial experience, mainly as programmers, but also as analysts, designers or maintainers (40% of the subjects did not have any experience at all). The subjects considered their knowledge of the UML to be high/very high in most cases for class diagrams and for sequence diagrams, and most of them had an excellent knowledge of JAVA language (all the subjects considered this to be above average). The grading system is analogous to that explained for E-US1.
- **R-UB**: The participants were 78 Computer Science students who were taking the Software Engineering course in the second-year of their Laurea Degree at the University of Bari. In Italy, the exam grades are expressed as integers and assume values of between 18 and 30. The lowest grade is 18, while the highest is 30. We cannot provide details about their background because the first session of the experiment needed to be canceled for reasons beyond our control.

The students who participated in the experiment and the replications were volunteers selected for convenience (the students available in the corresponding course).

4.3.3. Variable selection

The independent variable (also called the "main factor") is the *Origin* of diagrams, which is a nominal variable with two values (treatments):

- Forward Designed (FD)
- Reverse Engineered (RE).

The dependent variable is the maintainability. This dependent variable was measured using the following measures:

- **Maintainability Effectiveness (MEffec)**: This measure is related to the correctness of the response, and it therefore reflects the ability to maintain the system presented correctly. A higher value of this measure reflects better maintainability effectiveness. It is calculated with the following formula:

$$\text{MEffec} = \frac{\#\text{ correct tasks} - \#\text{ performed tasks}}{\#\text{ tasks}}$$

- **Maintainability Efficiency (MEffic)**: This measure is related to the timing of the response, but also reflects the ability to maintain the system presented correctly. Its unit of measure is "the number of correctly-performed modification tasks per time unit". The unit of time used was seconds. A higher value of this measure reflects better maintainability efficiency. It is calculated with the following formula:

$$\text{MEffic} = \frac{\#\text{ correct tasks} - \#\text{ performed tasks}}{\text{time spent}}$$

We also considered a further independent variable (from here on termed as the "cofactor"): *Ability*. We considered this cofactor in our efforts to investigate whether subjects' abilities play any role in the maintenance of source code, i.e., we discriminate between users according to their respective level of *Ability*, with the purpose of testing the hypothesis that this is a relevant influencing factor that should be taken into account when adopting these kinds of diagrams. A quantitative assessment of the participants' *Ability* was obtained by computing the final grade of the course they were taking. Those students from E-US1 and R-US2 with a final grade of below 5.7/10 (which represents the median of the group) were classified as low *Ability* participants, while those with a higher grade were given the classification of high *Ability* students. The instructor of the course (the last author of the paper), who was not one of the experimenters, was asked to provide the grades. In the case of R-UB, the cutoff point was 2.5/5.

4.3.4. Hypotheses formulation

The following hypotheses have been formulated and tested:

- $H_{1,0}$: There is no significant difference in the subjects' maintenance effectiveness when working with UML diagrams which have originated from the design phase or with diagrams which have originated from a Reverse Engineering technique. $H_{1,1}$:$\neg H_{1,0}$
- $H_{2,0}$: There is no significant difference in the subjects' maintenance efficiency when working with UML diagrams which have originated from the design phase or with those which have originated from a Reverse Engineering technique. $H_{2,1}$: $\neg H_{2,0}$

The goal of the statistical analysis is to reject the null hypotheses and possibly to accept the alternative ones. Both of the hypotheses are two-sided, because we did not postulate any effect arising from the origin of the diagrams.

4.3.5. Experimental design

When designing the experiment we attempted to alleviate several issues that might threaten the validity of the research done by considering the suggestions provided in (Wohlin et al., 2012).

We selected a between-subjects balanced design in which each treatment has an equal number of subjects (Kirk, 1995). We decided to use a between-subjects design

rather than a within-subjects design owing to time constraints. The inherent threats of a between-subjects design were thus alleviated by taking into account the suggestions provided in (Wohlin et al., 2012). In an attempt to alleviate experience effects, we provided the subjects with a background questionnaire (which can be downloaded with the rest of the experiment material) in the training session which took place before carrying out the experiment. The background questionnaire consisted of three blocks of questions. The first one was related to the subjects' skills and experience. The second block consisted of a set of question used to measure the subjects' knowledge of UML. The last block was a set of questions about Java. This background questionnaire improved on the design of this family of experiments in comparison with the previous experimentation work presented in the related work section.

The subjects were then assigned to the 2 groups in a random manner (see Table 4.3), based on the marks obtained in the background questionnaire (blocked design by experience). This was not possible in the case of the R-UB subjects, who were assigned to each group in a random manner.

To avoid skewing the results of the tasks as a result of their being of different levels of difficulty, they were randomized. The subjects in each group therefore received the same tasks but in a different order. In order to alleviate learning effects, the order of the tasks was the same for each treatment, i.e., one subject from each group received the tasks in the same order, but in a different order from the rest of his/her group.

Table 4.3. Experimental design.

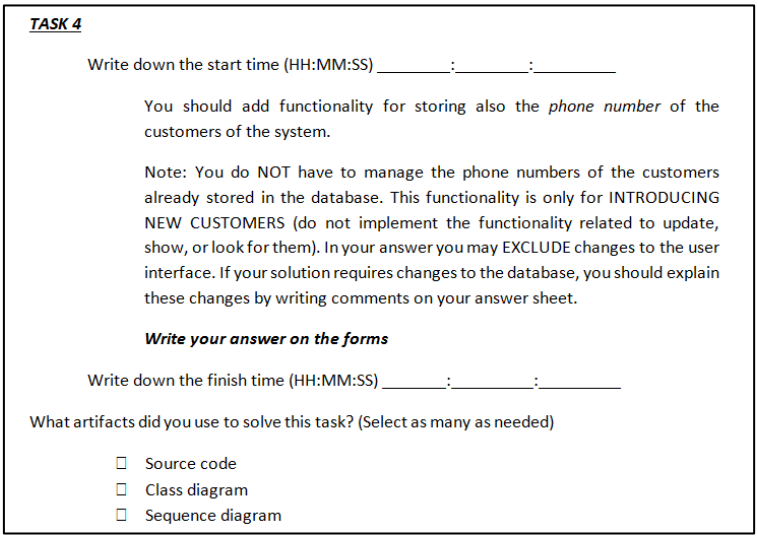
Origin of UML diagrams	
RE	*D*
Group 1	Group 2

4.3.6. Experimental tasks

The modification questionnaires were formed of two kinds of maintenance tasks (Table 4.4); both of these activities involve changing the source code:

- **Adaptive maintenance task**: these maintenance activities were intended to enhance the system by adding features, capabilities, and functions, in response to new technology, upgrades, new requirements, or new problems, i.e., a modification of a software product performed after delivery to keep a software product usable in a changed or changing environment (ISO/IEC, 1999). In our case, new requirements had to be added to the system, with the subjects receiving a list of requirements which had to be used to modify the code of the system and thus add/change certain functionalities. This part of the experiment consisted of 3 tasks (see example in Figure 4.2).
- **Corrective maintenance task**: these maintenance activities were "intended to remove errors or bugs from the software, the procedures, the hardware, the network, the data structures, and the documentation" (Swanson, 1976). In our case,

bugs from the source code had to be detected and fixed. We consequently analyzed the list of bugs reported by a professional Dutch IT development company (whose name is not shown for reasons of privacy) and introduced these kinds of defects into our system, giving the subjects a list of functional defects which had to be detected and corrected. All this explains why we consider these tasks to be common, realistic tasks. This part of the experiment consisted of 2 such tasks (see example in Figure 4.3). The subjects were provided with answer sheets for this kind of questions, to allow them to structure their responses.



TASK 4

Write down the start time (HH:MM:SS) _____:_____:_____

You should add functionality for storing also the *phone number* of the customers of the system.

Note: You do NOT have to manage the phone numbers of the customers already stored in the database. This functionality is only for INTRODUCING NEW CUSTOMERS (do not implement the functionality related to update, show, or look for them). In your answer you may EXCLUDE changes to the user interface. If your solution requires changes to the database, you should explain these changes by writing comments on your answer sheet.

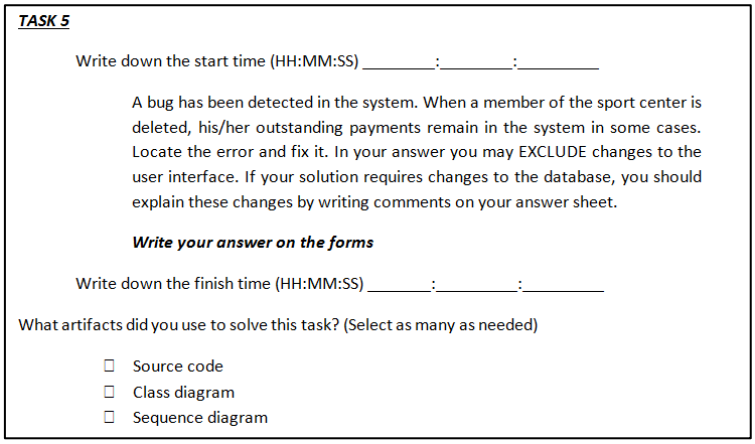
Write your answer on the forms

Write down the finish time (HH:MM:SS) _____:_____:_____

What artifacts did you use to solve this task? (Select as many as needed)

- ☐ Source code
- ☐ Class diagram
- ☐ Sequence diagram

Figure 4.2. Example of adaptive maintenance task.



TASK 5

Write down the start time (HH:MM:SS) _____:_____:_____

A bug has been detected in the system. When a member of the sport center is deleted, his/her outstanding payments remain in the system in some cases. Locate the error and fix it. In your answer you may EXCLUDE changes to the user interface. If your solution requires changes to the database, you should explain these changes by writing comments on your answer sheet.

Write your answer on the forms

Write down the finish time (HH:MM:SS) _____:_____:_____

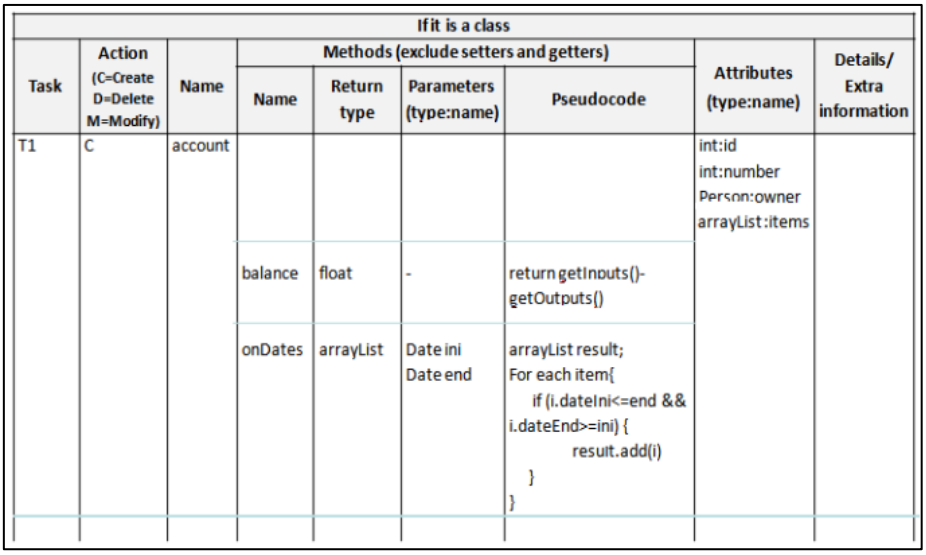
What artifacts did you use to solve this task? (Select as many as needed)

- ☐ Source code
- ☐ Class diagram
- ☐ Sequence diagram

Figure 4.3. Example of corrective maintenance task.

These two kinds of tasks needed to be answered using data collection forms, i.e., templates which had to be filled in with pieces of code (see example in Figure 4.4).

We used these data collection forms to obtain a structured response which facilitated the correction of the results, and is also an improvement on the design of this family of experiments in comparison with previous work. The subjects were provided with answer sheets to allow them to structure their responses regarding the maintenance tasks. The reason for doing this was that maintaining source code on paper is not easy owing to space constraints, and the subjects were therefore required to write changes to the source code in a structured manner on the answer sheets (format: line-no, change type, Java code, etc.).They had to fill in a different form depending on the element that they wished to maintain (a class, a method, an attribute, etc.). These answer sheets can be found at: http://alarcos.esi.uclm.es/originUMLmaintenance/



If it is a class

Task	Action (C=Create D=Delete M=Modify)	Name	Methods (exclude setters and getters)				Attributes (type:name)	Details/ Extra information
			Name	Return type	Parameters (type:name)	Pseudocode		
T1	C	account					int:id int:number Person:owner arrayList:items	
			balance	float	-	return getInputs()-getOutputs()		

			onDates	arrayList	Date ini
					Date end
						arrayList result;
						For each item{
						if (i.dateIni<=end && i.dateEnd>=ini) {
						result.add(i)
						}
						}		

Figure 4.4. Answer sheet related to the element "Class", filled with an example.

Table 4.4. Summary of maintenance tasks.

Task	Summary of task descriptions	Type of maintenance	Maximum mark
T1	When one of the sport center's services is not available (owing to a breakdown, for example) all reservations for this service should be cancelled.	Corrective	4 points
T2	The sport center's system should store its customers' telephone numbers.	Adaptive	5 points
T3	A ticket showing a customer's reservations at a specific time should be generated by the system.	Adaptive	5 points
T4	When we delete one of the sport center's members, his/her pending payments sometimes remain in the system.	Corrective	2 points
T5	The information about the sport center's instructors should be stored by the system.	Adaptive	6 points

The largest change consisted of adding a class which would need at least 22 lines of code. In general, between 1 and 3 classes needed to be modified. The complexity of the task might not appear to be too complex, owing to the number of LoCs that have to be changed, but the complexity of the task lies in the difficulty of detecting where the change is to be made in the source code, and how it should be carried out. It should also be borne in mind that 5 tasks had to be completed in 2 hours, using a system that the subjects had never seen. We limited the time of the experiment to fit in with subjects' availability. The subjects were only required to maintain the system, i.e., they did not need to update diagrams according to their changes or to create test cases.

After performing each maintenance task, the subjects were also required to indicate which artifacts (source code, class diagrams and/or sequence diagrams) they had used to solve the task. They were asked this in order to check whether or not they used the diagrams to solve the maintenance tasks (otherwise, the effect measured would not be the influence of the different origin of the diagrams). We shall refer to these kinds of questions as "post-questions".

Table 4.5. Post-Experiment Survey.

Id	Question/Issue	Possible Answers
Ex1	The difficulty of tasks	(1-5)
Ex2	The training was sufficient to be able to perform the tasks	(1-5)
Ex3	The clarity of the material provided	(1-5)
Ex4	The task objectives were perfectly clear to me	(1-5)
Ex5	The tasks I performed were perfectly clear to me	(1-5)
Ex6	I did not experience difficulty in reading the diagrams	(1-5)
Ex7	I did not experience difficulty in reading the source code	(1-5)
Ex8	The LoD of the diagrams was correct enough for me to be able to perform the tasks	(1-5)
Ex9	The available class diagrams were helpful	(1-5)
Ex 10	In the event that you do not think that the class diagrams have been useful, indicate why	Open question
Ex 11	The available sequence diagrams were helpful	(1-5)
Ex 12	In the event that you do not think that the sequence diagrams have been useful, indicate why	Open question
Ex 13	I had enough time to perform the tasks	Multiple choice question
Ex 14	How much time (as a percentage) did you spend looking at the diagrams?	Multiple choice question
Ex 15	How much time (as a percentage) did you spend looking at the source code?	Multiple choice question
1 = strongly agree; 2 = agree; 3 neutral; 4 = disagree; 5 = strongly disagree (Ex2, Ex3, Ex4, Ex5, Ex6, Ex7, Ex9, Ex11)		
1 = very high; 2 = high; 3 = correct; 4 = low; 5= very low (Ex8)		
1= very difficult; 2=difficult; 3=medium; 4=easy; 5=very easy (Ex1)		
1=very clear; 2=clear; 3=correct; 4=unclear; 5=very unclear (Ex3)		
A=more time needed; B=less time needed; C=enough time (Ex13)		
A. <20%; B. >=20% and <40%; C. >=40% and <60%; D. >=60% and <80%; E. >=80% (Ex14, Ex15)		

In addition, at the end of the experiment's execution the subjects were asked to fill in a post-experiment survey (see Table 4.5), whose goal was to obtain feedback about their perception of the experiment execution - feedback which could be used to explain the results obtained. The answers to the questions were based on a five-point Likert scale (Oppenheim, 2000). During the experiment's execution, the subjects had to perform 5 maintenance tasks, in different orders, which are summarized in Table 4.3.

4.3.7. Experimental procedure

The experimental material and the time duration were checked by carrying out a pilot study with 6 PhD students from the University of Castilla-La Mancha, Spain, before the execution of the first experiment. The pilot study was similar to the experiment described in this section, but with no time limit. The results of the pilot study were used as a basis to adapt the number of tasks and their complexity to the experimental time constraints. Some spelling mistakes were also corrected and some requirement statements were rewritten in order to make them more understandable.

We performed internal replications of the original experiment, and the entire family of experiments was carried out by the same experimenters. We conducted the experiment and the replications in a classroom under controlled conditions.

We did not provide details on the experimental hypotheses, and informed the participants that their grade on the course would not be affected by their performance.

The experiment and replications took place in two sessions of two hours each:

- Training session: The subjects first attended a training session in which detailed instructions on the experiment were presented and the main concepts of UML and JAVA were revised. In this session, we did not provide details on the experimental hypotheses and the subjects carried out an exercise similar to those in the experimental tasks in collaboration with the instructor. During the training session, the subjects were required to fill in a background questionnaire. The participants were informed that the data collected in the experiments were to be used for research purposes and would be treated as confidential, and that their grade on the course they were taking would not be affected by the grade obtained in the experiment. After the introductory lesson, we assigned the participants to one of the 2 groups in accordance with the marks obtained in the background questionnaire, thus obtaining balanced groups. Those subjects who did not come to the training session were randomly assigned to the groups, which in some cases led to unbalanced groups (a difference of 1 subject).
- Execution session: The execution of the experiment took place in the second session, in a classroom, where the students were supervised by the instructor of the course (a different one depending on the replication) and one experimenter (always the same one), and no communication between them was allowed. Each of the groups was given a different treatment.

After the execution of the experiment, the data collected from it were placed on an excel sheet, following an answering diagram constructed before the experiment was carried out. On this sheet, each task has a maximum mark (see Table 4.4), depending on the correctness of the answer provided. This means that for each task, a mark was given to the subject depending on the number of correct lines of code added to the solution. Incorrect answers were not given negative marks, i.e., lines of code which do not solve the task.

4.3.8. Analysis procedure

The data analysis was carried out by considering the following steps:

1. We first carried out a descriptive study of the measures of the dependent variable, i.e., MEffec and MEffic in order to obtain a general overview of the influence of the main factor (*Origin*).
2. We then analyzed the characteristics of the data to determine which parametric or non-parametric test it would be better to use. We performed a Kolmogorov-Smirnov test (Sheskin, 2007) to determine the normality of distributions and a Levene (Sheskin, 2007) test to determine the homogeneity of variances. These analyses are useful in determining which parametric or non-parametric test it is best to use.
3. Based on the results of the previous test, for the data collected in each experiment we tested the hypotheses formulated using the parametric ANOVA test (Devore and Farnum, 1999) when the results of Kolmogorov-Smirnov test and Levene test were positive. The non-parametric Mann-Whitney test (Sheskin, 2007) was performed when the data obtained did not satisfy the restrictions of the ANOVA test (we did not obtain normal distributions, and there is no homogeneity of variances).
4. To strengthen the results of each experiment, we decided to integrate them using a meta-analysis. A meta-analysis is a set of statistical techniques which are used to combine the different effect sizes of the experiments in order to obtain a global effect of a factor in a dependent variable.
5. We analyzed the influence and the interaction of the cofactor, i.e., *Ability*, using statistical tests. The interaction of the *Ability* with the main factor (i.e., *Origin*) was also tested using interaction plots (Devore and Farnum, 1999). Interaction plots are simple line graphs in which the means on the values of a dependent variable for each level of one factor are plotted on all the levels of the second factor. The resulting lines are parallel when there is no interaction and nonparallel when an interaction is present.
6. The data collected from the post-experiment survey were eventually analyzed using bar graphs. In cases in which a pattern was detected in the data, these data were also tested with a T-test (Sheskin, 2007) owing to their nature.

All the statistical values were calculated using SPSS (SPSS, 2003), with its standard configuration. In all the statistical tests, we decided to accept a probability of 5% of committing a Type-I-Error (Sheskin, 2007).

4.3.9. Documentation and communication

Issues such as documentation (Shull et al., 2004) and communication among experimenters (Vegas et al., 2006) may influence the success or the failure of an experiment performance and its future replications. Laboratory packages and knowledge-sharing mechanisms were used to handle these issues. The material was originally written in Spanish for E-US1 and R-US2, and was then translated into Italian for R-UB. The material included: the post-experiment survey, the modification questionnaires, the data collection forms, the source code and the UML diagrams (two versions: D and RE). The groups of experimenters also shared a document to provide a common background so as to be able to communicate all terms related to the design and analysis of the experiment.

The experimenters (the first three authors of the paper) began with an initial face-to-face meeting in which the main ideas of the experiments were discussed and reported in an agreement document. All the experimenters then exchanged the agreement documents from the meeting by e-mail to attain a shared common research plan. This phase played a significant role in sharing knowledge among the experimenters and in the discussions on possible issues related to the study that might arise.

The experimenters used instant messaging tools and e-mails to establish a communication channel in all phases of the study. Teleconferences were also held to share knowledge among the research groups and to discuss the experimental procedure that the participants had to follow.

4.4. RESULTS

In this section, we present the data analysis following the procedure presented above: the presentation of the descriptive statistics, the test of the hypotheses related to the main factor (*Origin*), the analysis of the influence of the cofactor *Ability* and the analysis of the post-experiment survey.

Please bear in mind that in some cases the groups are not balanced (there are more subjects in one group than in the other) owing to the fact that some subjects abandoned the experiment during its execution.

4.4.1. Descriptive statistics and exploratory analysis

Table 4.6 and Table 4.7show the descriptive statistics of the Maintainability measures, i.e, MEffec and MEffic respectively, grouped by the *Origin* of the UML diagrams. These tables contain the following data: number of subjects (N), mean ($\overline{X}$), median, and standard deviation (SD).

At a glance, we can observe that when the subjects used FD diagrams they obtained better values (although the difference is very low in both measures when comparing means (only in the case of MEffec in R-UB are the results are better with RE diagrams). This indicates that there is a slight tendency towards FD diagrams improving the performance of software code maintainers.

Table 4.6. Descriptive statistics for MEffec.

Origin	MEffec							
	RE				FD			
	N	$\overline{X}$	Median	SD	N	$\overline{X}$	Median	SD
E-US1	20	0.641	0.6818	0.165	20	**0.650**	0.6818	0.148
R-US2	23	0.597	0.5909	0.223	28	**0.667**	0.6818	0.174
R-UB	39	**0.512**	0.5000	0.171	39	0.441	0.4091	0.219

Table 4.7. Descriptive statistics for MEffic.

Origin	MEffic							
	RE				FD			
	N	$\overline{X}$	Median	SD	N	$\overline{X}$	Median	SD
E-US1	20	0.00270	0. 00283	0.00079	20	0.**00273**	0. 00303	0.00072
R-US2	22	0.00261	0.00245	0.00106	28	**0.00350**	0.00352	0.00147
R-UB	39	0.00217	0.00204	0.00087	39	**0.00255**	0.00183	0.00406

4.4.2. Influence of Origin of diagram

In order to test the formulated hypotheses ($H_{1,0}$, $H_{2,0}$) we analyzed the effect of the main factor (i.e. *Origin*) on the measures considered (i.e., MEffec and MEffic) using the non- parametric Mann-Whitney or ANOVA test, depending on the normality of data (as explained in Section 4.3.7).

In the following subsections, the results for each measure of the Mann-Whitney U tests or ANOVA tests are shown in tables (Table 4.8, and Table 4.9), in which the *Origin* column describes the independent variable, *p-value* is the statistical significance obtained, *op* is the estimated observed power of the test, *es* is the effect size, and *R* describes whether the data obtained allows us to reject the null hypothesis, while the tendency of the data in the case of the null hypothesis being rejected is shown in the "*in favour of…*" column.

The results obtained for each hypothesis will be commented on in their corresponding subsections.

For each measure, we first decided to analyze the data related to maintenance in general, as is presented in the formulated hypothesis. We then made the decision to analyze the results by dividing them by the type of maintenance, since there may have been differences between the results from the adaptive and the corrective maintenance.

4.4.2.1. Testing Maintenance Effectiveness: MEffec ($H_{1,0}$)

The first row of Table 4.8 shows that we cannot reject $H_{1,0}$ in the first experiment (E-US1) and the first replication (R-US2) given that their p-value is greater than 0.05. Hence, here the different origins of UML diagrams had no effect on the subjects´ effectiveness when performing the source code maintenance tasks. The observed power of the test is low, probably because of a small effect size, so we assume a 0.946 (1 - 0.054) and a 0.755 (1 – 0.245) estimated probability of a Type II error in our

assertions. Given the low value of the observed power, we cannot obtain strong conclusions from E-US1 and R-US2.

In the case of the second replication, the p-value is 0.047, i.e., lower than 0.05, and we can therefore reject the null hypothesis. In this case we can assume that there is a difference in the maintenance effectiveness when using RE diagrams or FD diagrams in favor of RE diagrams. What is more, the power of the test is quite high in this case but it is still not sufficient for us to be confident of the results, because there is a probability of 65% of committing a Type II error in our assertion.

Table 4.8. Statistical relation between Origin of diagram (RE/FD) and Maintainability Effectiveness (MEffec).

Origin	MEffec					
	Test	p-value	op	es	R	In favour of…
E-US1	Man-Withney	0.957	0.054	0.001	NO	-
R-US2	ANOVA	0.202	0.245	0.033	NO	-
R-UB	Man-Withney	**0.047**	0.352	0.033	**YES**	**RE diagrams**

We also performed an analysis of the influence of the Origin on maintenance effectiveness per type of maintenance, i.e., adaptive and corrective maintenance. The results were not significant in all the cases because the p-value is always greater than 0.05.The same occurred when the MEffec as regards the time spent maintaining the system (without relating this to the number of correct answers) was tested.

4.4.2.2. Testing Maintenance Efficiency: MEffic ($H_{2,0}$)

The numbers in Table 4.9 show that in the case of E-US1 and R-UB there are no significant effects (the p-values are not smaller than 0.05) as regards the *Origin* of UML diagrams on maintenance efficiency and that, in this case, the statistical power is still very low. But, if the null hypothesis were to be accepted, we would be assuming a 0.949 (i.e., 1-0.051) estimated probability of a Type II error.

But as was the case with MEffec, there is one case in which there is an influence of the Origin on the maintainability efficiency in favor of FD diagrams. In this case, in R-US2 the p-value is 0.049, and the null hypothesis is therefore rejected.

Table 4.9. Statistical relation between Origin of diagram (RE/FD) and Maintainability Efficiency (MEffec).

Origin	MEffic					
	Test	p-value	op	es	R	In favour of…
E-US1	Man-Withney	0.534	0.051	0.0003	NO	-
R-US2	Man-Withney	**0.049**	0.343	0.049	**YES**	**FD diagrams**
R-UB	Man-Withney	0.272	0.088	0.004	NO	-

Once again, an analysis of the influence of the *Origin* on maintenance efficiency per type of maintenance, i.e., adaptive and corrective maintenance, was performed. In this case, the results were not significant either (the p-values are greater than 0.05).

We also attempted to measure MEffic as regards the time spent maintaining the system, without relating this to the number of correct answers (as was done before). In

this case, the p-value was again higher than 0.05 (i.e., p-value=0.725) but with a higher statistical power (i.e., op=0.5). The same result was obtained in the test with the MEffic measure as regards the time spent maintaining the system (without relating this to the number of correct answers).

4.4.2.3. Integrating the obtained results through meta-analysis

When the different effect sizes of the experiments need to be combined to obtain a global effect of a factor, the statistical technique used is that of meta-analysis. Although we have found some significant results: in favor of RE diagrams in relation to MEffec in R-UB, and in favor of FD diagrams in relation to MEffic in R-US2, there are no significant results in the remaining cases. We have therefore decided to integrate the results of the different studies through a meta-analysis (in which the factor is the Origin of UML diagrams and how this affects the modifiability of source code), in order to explore if stronger results can be found. This technique has been used for the same purpose in other families of experiments, such as that shown in (Cruz-Lemus et al., 2010).

For each dependent variable, we computed the mean value obtained by the participants when using RE diagrams, minus the mean value they obtained with FD diagrams, and these values were then used to compute the Hedges' g metric (Hedges and Olkin, 1985; Kampenes et al., 2007). The overall conclusion was obtained by calculating the Z score based on the mean and standard deviation of the Hedges' g statistics of the experiments. The global effect size was therefore obtained by using the Hedges' g metric, with the weights proportional to the experiment size:

$$\overline{Z} = \frac{\sum_i w_i z_i}{\sum_i w_i}$$

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 FD diagrams is half a standard deviation larger than the mean value obtained when not using them.

As suggested in (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 meta-analysis was performed by using the Comprehensive Meta-Analysis v2 tool (Biostat Inc., 2006). For each measure, the tool produced the forest plots depicted in Figure 4.5 and Figure 4.6. The squares and diamonds are mostly proportional in size to each study's weight under the fixed effect model (see the 'Relative weight' column). The squares show the individual effect size of each experiment and the diamond shows the global effect size. The values of the Hedges' g metric are also reported. Positive values indicate that the use of FD diagrams improves the modifiability of source code. Negative values signify that RE diagrams are the best treatment.

If we focus on the results obtained for the MEffec variable (see Figure 4.5), the total effect is in favor of using FD diagrams, but the global effect size obtained is not statistically significant since the p-value is greater than 0.05. The value obtained for the Hedge's g metric, i.e.,0.282, indicates a small size for the global effect. Similar results were obtained for MEffic (see Figure 4.6), where the total effect is again in favor of FD diagrams, but the size of the effect is small. In both cases, if the coincidence interval were to be increased to 90% instead of 95%, then the meta-analysis would be significant because the p-values would not be greater than 0.10.

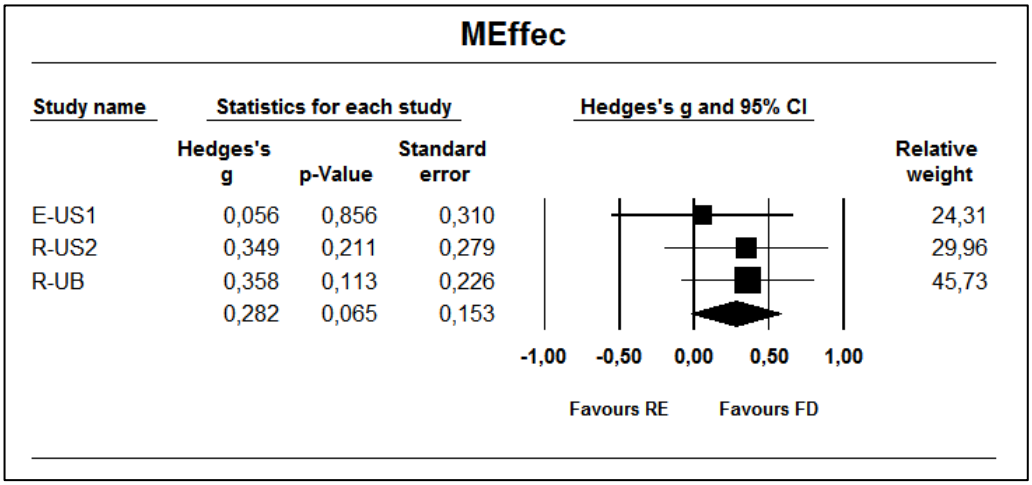


Figure 4.5. Meta-analysis results for Maintainability Effectiveness (MEffec).

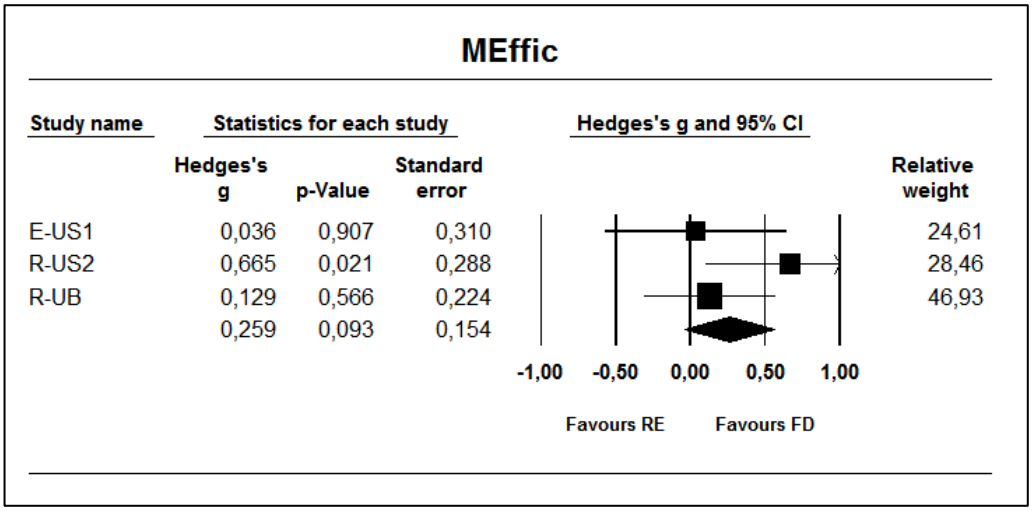


Figure 4.6. Meta-analysis results for Maintainability Efficiency (MEffic).

4.4.3. Influence of Ability

We tested whether the subjects' *Ability* influenced the results but. The effect of this factor could not be confirmed because the p-value found was higher than 0.05 in all of the cases (see Table 4.10), i.e., the subjects' *Ability* had no statistical influence on the results. This was expected owing to the balanced design of the experiment.

The interaction plot shown in Figure 4.7 indicates that there was no interaction between *Origin* and *Ability* for MEffec. In the case of E-US1 and R-US2, high ability participants achieved better scores than low ability ones when both of them were using RE and FD diagrams. The interaction plot also suggests that the results achieved with FD diagrams are better than those obtained with RE diagrams for both high and low ability participants. This might be owing to the fact that RE diagrams contain too many details when compared with FD diagrams. In particular, RE sequence diagrams are twice as large in terms of messages when compared to FD diagrams. This could be because FD diagrams only contain logical messages between objects, and that messages between other kinds of objects are obviated, such as objects from Java packages, which are shown in RE diagrams. This difference between RE and FD diagrams is based on their nature, owing to the fact that human based diagrams contain less technical details than RE diagrams. In the case of R-UB, the effect of the combination of the Ability and the Origin is contrary to the other cases, i.e., the results achieved with RE diagrams are better than those obtained with FD diagrams for both high and low ability participants.

Table 4.10. Statistical relation between Ability and Maintainability Effectiveness (MEffec) and Efficiency (MEffic).

Ability	MEffec	MEffic
E-US1	0.226	0.914
R-US2	0.460	0.939
R-UB	0.470	0.995

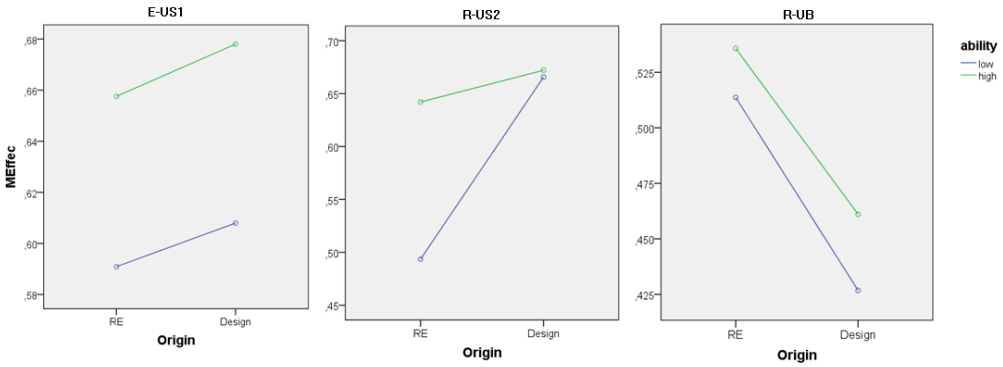


Figure 4.7. Interaction between Origin and Ability for Maintainability Effectiveness (MEffec).

As with MEffec, the influence of the subjects' *Ability* and the *Origin* in the case of MEffic was also represented on an interaction plot. The interaction plot shown in

Figure 4.8 indicates that there was a clear interaction between *Origin* and *Ability* for MEffic. In this case, high *Ability* participants achieved better scores using the FD diagrams, and low *Ability* participants did better using the RE diagrams (except in the case of R-US2). This might be explained by the fact that RE diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams. However, experienced maintainers do not need very high traceability, because using FD diagrams might allow them to obtain sufficient information to attain a correct overview of how the system works.

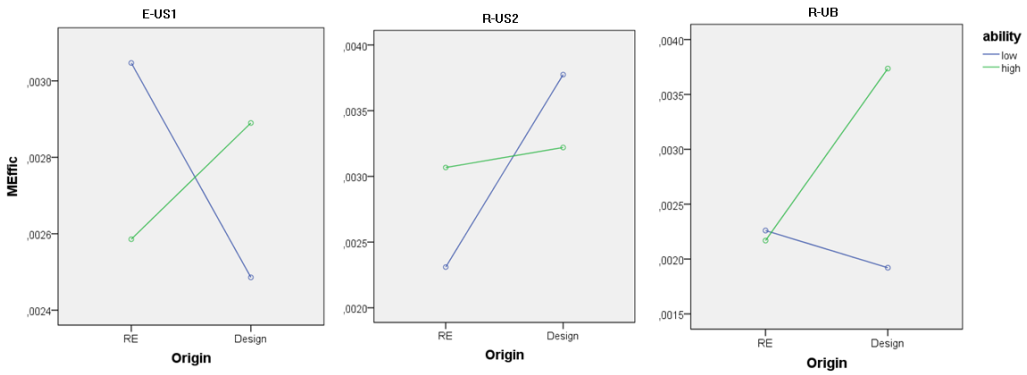


Figure 4.8. Interaction between Origin and Ability for Maintainability Efficiency (MEffic).

4.4.4. Post-experiment survey results

The analysis of the answers to the post-experiment survey revealed that the time needed to carry out the modification tasks (Figure 4.9) was not considered to be sufficient (more time was needed), and that the subjects considered that the performance of the tasks was of more or less low-medium difficulty (Figure 4.10), independently of the particular treatment received. The need for more time to perform the tasks may have arisen from the fact that the measurement of the time needed was derived from the pilot study, which was performed by PhD students, who probably had higher ability and/or more experience than these Master's students, signifying that the less experienced subjects needed more time. We would also like to state that some subjects did not finish the questionnaire owing precisely to this lack of time.

We also asked about the subjects' perception of the adequacy of the LoD of the diagrams used. The majority of the subjects who received FD diagrams agreed with the LoD of the diagrams they received (i.e., they considered the LoD as the "right" amount of detail). In the case of those subjects who received RE diagrams, the majority agreed with the LoD presented, but of those subjects who did not, the majority required more LoD (Figure 4.11).

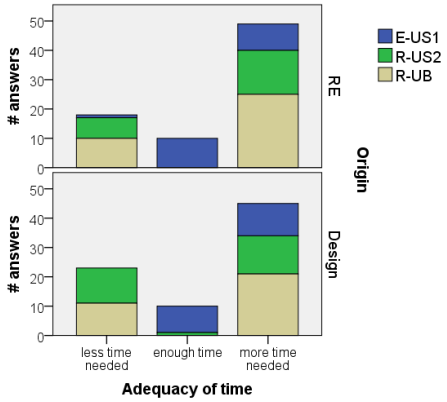


Figure 4.9. Subjects' answers as regards adequacy of time provided.

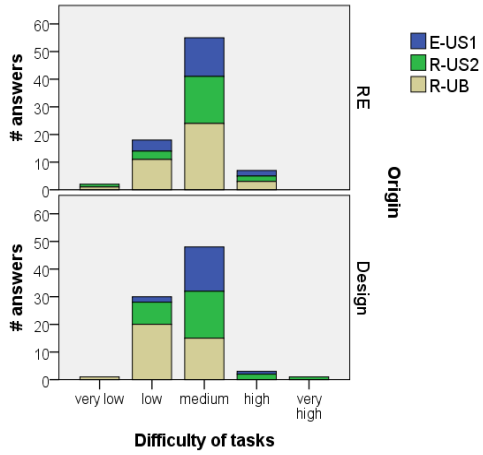


Figure 4.10. Subjects' answers as regards difficulty of task.

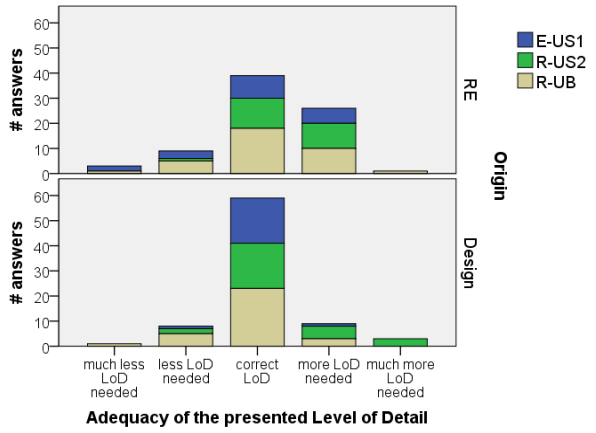


Figure 4.11. Subjects' answers as regards adequacy of the LoD.

The subjects who received FD diagrams experienced fewer difficulties when reading the diagrams used, in comparison with the RE group, as is shown in Figure 4.12. We tested whether there was a difference as regards the difficulties experienced by subjects depending on the diagrams they used. This was done by using a Mann-Whitney test owing to the nature of the data, and this test was carried out by comparing the subjects' responses (from 1=very low to 5=very high) grouped by the UML diagrams that they had used (RE or FD diagrams). The results of the test show a significant difference (p-value=0.007, which is lower than α=0.05). The power of the test is high (0.743), and this therefore allows us to state that the subjects who received RE diagrams experienced more difficulties when reading diagrams than those who received FD diagrams.

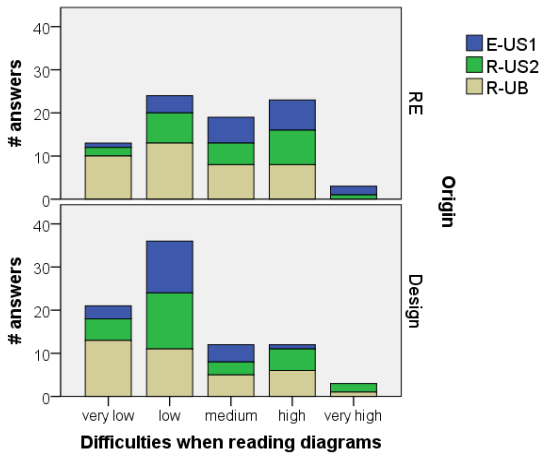


Figure 4.12. Subjects' answers as regards to difficulties when reading diagrams.

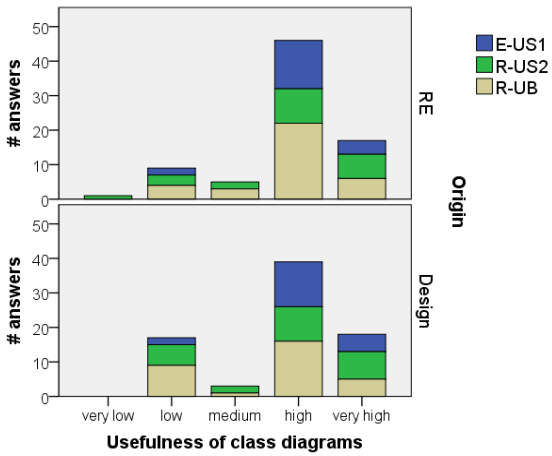


Figure 4.13. Subjects' answers as regards usefulness of class diagram used.

As part of the post-experiment survey, the subjects were required to indicate how useful the diagrams were for them in general as regards solving tasks. Class diagrams are considered useful in both groups, in more or less the same proportion (Figure 4.13). Having said that, the majority of the subjects who received the RE diagrams commented that the sequence diagrams employed were not useful and were very difficult to understand, as opposed to the majority of the subjects in the D group who considered them helpful (Figure 4.14). This finding may have been caused by the different complexities and varying LoD in the different kinds of diagrams, as explained in previous sections.

A Mann-Whitney test was then used (owing to the nature of the data) to test whether there was a difference as regards the subjects' perceived usefulness of the sequence diagrams depending on the diagrams they used. The test was carried out with the same configuration as explained in the test for the difficulties when reading the diagrams. The results of the test again show a significant difference (p-value=0.002, which is lower than α=0.05). The power of the test is high (0.88), and this therefore allows us to state that the subjects who received RE sequence diagrams considered them to be less helpful than the sequence diagrams in comparison to those who received FD sequence diagrams.

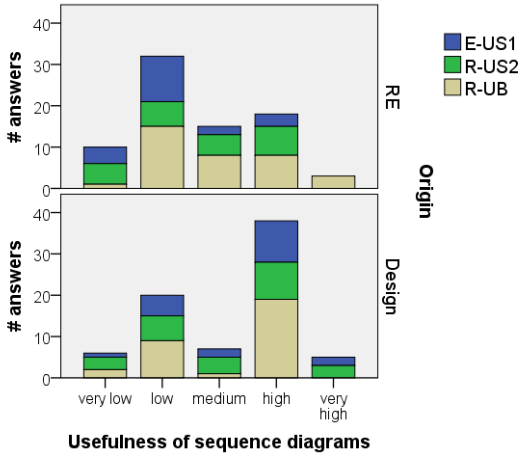


Figure 4.14. Subjects' answers as regards usefulness of sequence diagrams used.

The results of the post-questions concerning the artifacts used to answer the questions led us to detect that almost all the subjects used the source code to solve the tasks. This was expected, in the sense that source code is needed when it is being maintained. The use of the source code was from 7 to 25% higher in the RE group than in the D group.

We then analyzed whether or not the subjects had used the diagrams (Table 4.11). Class diagrams were also used by the majority of subjects (except in the case of the D group in R-UB). These percentages are consistent with the subjective response

provided in the post-experiment survey (see Figure 4.13). In the case of the RE group, the subjects used class diagrams in more or less the same proportion for corrective or perfective tasks, but in the case of the D group, the subjects used from 7 to 27% more class diagrams for perfective tasks. This may have occurred because class diagrams provide the structure of the system, thus allowing maintainers to obtain an overview of the system faster, which would appear to be easier with the FD diagrams owing to their conciseness; this is more important for perfective tasks. If we focus on the use of sequence diagrams, we would like to highlight that their use was surprisingly low; in general, only 20 to 41% of the subjects used them. That is consistent with the RE group subjects' opinions (Figure 4.14), in which they indicate that they did not use sequence diagrams, and they also believe that they are not useful diagrams for understanding the system during its maintenance. In the case of the D group, there is an inconsistency arising from the fact that the subjects did not use sequence diagrams in most of the tasks, even though they considered them to be useful (see Figure 4.14). Subjects from both groups (except the RE group of R-US2) used the sequence diagram more in corrective tasks than in perfective tasks (a difference of 2 to 50%). The reason for this could be that for corrective tasks, in which maintainers need to localize an error, structure and behavior are needed, since the error might be caused by a structural error or by a behavior error.

Table 4.11. Usage of UML diagrams for solving tasks.

Exp	Usage of class diagrams		Usage of sequence diagrams	
	RE	**D**	**RE**	**D**
E-US1	80%	74%	33%	33%
R-US2	59%	69%	23%	29%
R-UB	67%	40%	41%	20%

4.4.5. Summary and discussion of the data analysis

The experiment (E-US1) was performed with 40 students from the University of Seville (Spain), and it was replicated in the same university with 51 students (R-US2) and in the University of Bari, Italy, with 78 students (R-UB).

Descriptive statistic results show that subjects using FD UML diagrams obtained better values in both measures (except in the case of MEffec in R-UB), indicating that FD diagrams may, to some extent, improve the maintenance of the source code, but that the differences are very slight.

As regards the results of the statistical test, in almost all of the cases the variables (i.e., MEffec and MEffic) are not significantly affected by the *Origin* of the UML diagrams, i.e., the results of the tests performed did not allow us to reject any of the null hypotheses presented in section 4.3 in almost all cases, as all the significance levels are above 0.05. The test powers are low, so the possibility of an error occurring as a result of accepting the null hypothesis is high. But if we focus on the case of R-US2, the results of the MEffic were influenced by the Origin, thus obtaining a high

test power. The same occurred in the case of R-UB with MEffec, where there is a clear difference in favor of FD diagrams.

Despite the fact that the results were not conclusive in all the cases (which is not as positive as we expected), we have ensured that the experimental results were not influenced by other cofactors such as the subjects' *Ability*. If we focus on the interaction between *Origin* and *Ability*, in the case of E-US1 and R-US2, the subjects using FD diagrams achieved better results than those who used RE diagrams, in the case of both high and low ability participants. This may have been caused by the fact that RE diagrams contain too many details when compared with FD diagrams. In the case of R-UB, the effect of the combination of the Ability and the Origin is contrary to the other cases, i.e., the results achieved with RE diagrams are better than those obtained with FD diagrams in the case of both high and low ability participants. The low ability users obtained more benefits from RE diagrams than from FD diagrams in terms of efficiency owing to the high traceability between RE diagrams and code.

The results of the family of the experiments are summarized in Table 4.12.

Table 4.12. Summary of the results of the family of experiments.

Exp_ID	Descriptive statistics (in favour of…)		Influence of Origin		Influence of Ability	
	MEffec	MEffic	MEffec	MEffic	MEffec	MEffic
E-US1	FD	FD	✘	✘	✘	✘
R-US2	FD	FD	✘	✓	✘	✘
R-UB	RE	FD	✓	✘	✘	✘

Legend

Descriptive statistics:

"FD" = better results when using **FD** diagrams than RE diagrams

"RE" = better results when using **RE** diagrams than FD diagrams

"-" = **no differences** when using FD or RE diagrams

Influence of Origin:

✘ = hypothesis not rejected →There is **no** significant difference in the results when working with FD or RE UML diagrams

✓ =hypothesis rejected →There **is** a significant difference in the results when working with FD or RE UML diagrams (**expected value**)

Influence of Ability:

✘ = hypothesis not rejected →There is **no** significant difference in the results when working with high or low ability students (**expected value**)

✓ =hypothesis rejected →There **is** a significant difference in the results when working with high or low ability students

After the individual analysis, a meta-analysis was performed in order to clarify the results. Its results also show a tendency in favor of FD diagrams for both measures (MEffec and MEffic). But in both cases, the results are not clearly evident owing to the values of the p-values.

Moreover, if we study the results of the post-experiment survey, we can see better subjective results for the FD diagrams. This is because the subjects who received RE diagrams did not believe their sequence diagrams to be useful, since they were not understandable. Significant results were obtained, showing that the subjects who received RE diagrams experienced more difficulties when reading the diagrams used; this is especially true as regards sequence diagrams. In the case of the D group, there was an inconsistency arising from the fact that the subjects did not use sequence diagrams in most of the tasks, even though they considered them to be useful.

We would like to underline that, according to the results of the post-question for each maintenance task concerning the use of artifacts in task solving, UML class diagrams were used in more or less in the same proportion as the source code for system understanding during the maintenance tasks. The sequence diagram was less widely-used, probably because of the nature of the tasks presented during the course of this experiment (more perfective maintenance tasks were required than corrective maintenance tasks). As stated previously, UML diagrams are not usually updated during maintenance tasks owing to the time constraints in realistic environments. But the high level of use of class diagrams during this experiment leads us to recommend companies to keep them up to date in order to improve maintainers' performances.

4.5. IMPLICATIONS OF THE STUDY

No evident overall pattern emerges from the family of experiments of this study – especially when limiting to the statistics of the experiments: when considered individually 1 out of 3 experiments does not provide statistically significant results in favour of either RE or FD diagrams.

The experiments showed that high Ability participants achieved better scores using the FD diagrams, and low Ability participants did better using the RE diagrams (except in the case of R-US2). This might be explained by the fact that RE diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams. This group may find it difficult to make mentally bridge the gap between source code and high level class diagram. However, experienced maintainers do not need very high traceability, because using FD diagrams might allow them to obtain sufficient information to attain a correct overview of how the system works.

The key characteristics of RE and FD diagrams that we compare in this study are the following: RE diagrams are complete and close to source code. Man-made FD diagrams are almost never complete, but their strength seems to be the selective inclusion of information about the system and modest complexity of the diagrams.

Findings from the post-experiment survey show that subjects who received FD diagrams experienced fewer difficulties when reading the diagrams compared to the RE group (p-value=0.007), especially with the class diagrams. At the same time, RE sequence diagrams were found less useful than FD sequence diagrams (p-value=0.002). A possible explanation for this is that human engineers make (implicit)

assessments regarding which information is important for understanding key aspects of the system and thus regarding which information should be included in a diagram. We believe the findings of our experiments indicate that FD class diagrams provide a more attractive balance between detail and relevant information than RE class diagrams. For sequence diagrams, it seems that current automated reverse engineering methods are not able to filter out the relevant information for maintenance tasks.

On the other hand, for performing the maintenance tasks in our study, participants found that source code source to be the most useful source of information, closely followed by the class diagrams. The participants therefore perceived that UML diagrams might add little value over access to the source code for the tasks in our study. This result is relevant for the researcher because it might be interesting to investigate the motivation guiding a software engineer when trusting in a source of information and how he/she exploits it to accomplish a maintenance task.

4.6. THREATS TO VALIDITY

We must consider certain issues which may have threatened the validity of the experiment (Wohlin et al., 2012):

- **External validity**: External validity can be threatened when experiments are performed with students, and the representativeness of these subjects may be doubtful in comparison to that of software professionals. In spite of this, the tasks to be performed did not require high levels of industrial experience, so we believe that this experiment could be considered appropriate, as it follows suggestions made in the relevant literature (Basili and Weiss, 1984; Basili et al., 1999; Höst et al., 2000). Working with students also implies various advantages, such as the fact that their prior knowledge is fairly homogeneous, there is the possible availability of a large number of subjects (Verelst, 2004), and there is the opportunity to test experimental design and initial hypotheses (Sjøberg et al., 2005). An additional advantage of using novices as subjects in experiments on maintainability is that the cognitive complexity of the objects under study is not hidden by the subjects' experience. Nevertheless, it would be extremely interesting to carry out further replications of the experiment with practitioners.
- Another threat to external validity concerns the experimental material used. The selected system is representative of a small industrial system. In addition, the experimental objects are small. The results of the experiments might be different when using bigger systems and experimental objects. The size of the experimental objects could also threaten the external validity of the results. The rationale for selecting the experimental objects used relies on the need (owing to time constraints) to simulate actual maintenance tasks related to small maintenance operations that novice software engineers and/or junior programmers may perform in a software company.
- **Internal validity:** Threats to internal validity were mitigated by the design of the experiment. Each subject was grouped by his/her results in the background

questionnaire, so both groups had subjects with a similar skill level. In all the cases, the subjects' knowledge of UML and JAVA was reinforced by teaching them about UML diagrams and JAVA in a training session organized to take place the day before the experiment was carried out, but their knowledge was sufficient for them to understand the given system, and they all had roughly the same background (which was tested with a background questionnaire).

- Furthermore, all the participants found the material provided, the tasks, and the goals of the experiment to be clear, as the post-experiment survey results showed. Another safeguard was that the instrumentation was tested in a pilot study, to check its validity. In addition, mortality threats were mitigated by offering the subjects the possibility of performing similar tasks in the final exam of the course that they were taking. Another issue that is a potential threat is the exchange of information among the participants. We must emphasize that the participants were not allowed to communicate with each other, and were prevented from doing so by being monitored during the run of the experiment. When the experiment had concluded, the participants were asked to give back all the experimental material.
- **Construct validity**: This validity may be influenced by the measures used to obtain a quantitative evaluation of the subjects' performance, the maintenance tasks, and the post-experiment survey, in addition to social threats. We performed the experiment in a really short period of time owing to the subjects' constraints. The small amount of time that the students were given to perform the tasks could have influenced the results of this experiment, as could the small number of tasks, which was again owing to constraints on our subjects' time. The measures used were selected to achieve a balance between the correctness and completeness of the answers, which are well-known measures and are widely-used in this kind of experiments. The questionnaires were defined to obtain sufficiently complex questions, without them being too obvious. The post-experiment survey was designed using standard forms and scales. Social threats (e.g., evaluation apprehension) have been avoided, since the students were not graded on the results obtained. Absenteeism was avoided by performing similar tasks to the exercises that would appear in their final exam.
- Based on the results of these pilot study, we consider that the two experimental objects fit the time constraints of the experiment in that they are sufficiently realistic for small maintenance operations that novice software engineers perform within a software company (Cohen et al., 2004).
- The experimental material delivered to the subjects consisted of the UML diagrams (class and sequence diagrams) and the JAVA source code of the system, the answer sheets, and the post-experiment survey.
- **Conclusion validity**: Conclusion validity concerns the data collection, the reliability of the measurement, and the validity of the statistical tests, all or any of which might affect the ability to draw a correct conclusion. Statistical tests were used to reject the null hypotheses, but the fact that subjects performed a small number of tasks provided us with few data points to work with. These particular

statistical tests were selected by checking that they followed the specific assumptions related to their use. We have explicitly mentioned and discussed all those cases in which non-significant differences were present.

4.7. CONCLUSIONS AND FUTURE WORK

As software maintenance takes up a major part of software projects, we are interested in investigating which documentation is more helpful in supporting maintainers when they have to maintain source code. This paper specifically presents a family of experiments that were carried out to investigate whether using either FD or RE UML diagrams (class and sequence diagrams) improve the maintainer's performance when modifying source code. We therefore wished to ascertain whether it is beneficial to build UML diagrams in the initial phases of development and kept them up-to-date (which consumes more effort) or whether on the contrary it is better to rely on UML diagrams obtained by reverse engineering the source code.

The original experiment (E-US1) was carried out by 40 second-year students on a Master's Degree in Computer Science. The first replication (R-US2) was conducted by 51 students enrolled on the same degree and at the same university as the experiment. The second replication (R-UB) was performed with 78 second-year Laurea Degree students in Computer Science at the University of Bari.

The statistical results, and specifically the statistical test and the descriptive results, show a tendency towards obtaining better results when using UML diagrams (concretely class diagrams), that were hand-made during the design phase. Based on the qualitative results of the post-experiment survey, it is also important to note that the subjects preferred FD diagrams when understanding and maintaining a system. This is true even though their performance is not much better with FD diagrams in some cases, in comparison to their performance with RE diagrams. Because software maintenance is a human-based process, this highlighting of maintainers' perceptions in favor of using FD diagrams is very important.

In addition, we found that subjects who received RE diagrams experienced more difficulties when reading the diagrams, especially the sequence diagrams. Although the subjects who received FD diagrams felt that sequence diagrams were highly useful, as they expressed in the post-experiment survey, only a small number of subjects actually used the diagrams for the tasks in the experiment. In the case of the RE diagram group, the subjects did not use RE sequence diagrams. These subjects point out that these RE sequence diagrams are not very useful due to their low level of readability/high level of details. Even though the results of the family of experiments are not homogenous in all the experiments, the evidence shown in the MEffic test in R-US2, the results of the meta-analysis and the subjects' opinion extracted in the post-questionnaire survey point in the direction that software maintainers prefer to use FD UML diagrams over Reverse Engineered UML diagrams.

These results give us grounds to encourage software developers, albeit with caution, to follow a model-centric approach in projects with novel maintainers and

small-sized systems related to well-known domains. Nonetheless, other contexts should be explored in order to reaffirm the results in an industrial context by carrying out further replications with professionals, considering more complex systems related to non-well known domains.

The findings obtained imply beginning the development of a software system by creating UML diagrams, and in addition to keep them up-to-date, thereby making it easier to perform maintenance tasks. Class diagrams are important artifacts which are widely used and highly appreciated by maintainers. However, there is a doubt as to whether the documentation, and UML as part of it, will be maintained as the system evolves when a model-centric approach is used (Petre, 2013). This goes against an effective use of the diagrams, a fact that suggests that we should recommend companies to keep their diagrams up-to-date and thus help their maintainers to perform the required tasks efficiently. The results obtained are therefore useful for all those companies that exploit this notation as a support for software maintainers when performing maintenance tasks.

The recommending on the use of Forward Designed UML diagrams (at least the class diagrams) assumes that UML diagrams were created during software development, and kept up-to-date during software maintenance. This requires an additional effort in the software lifecycle in comparison with the automated generation of UML diagrams from source code through reverse engineering techniques. Currently it seems that reverse engineering techniques are not able to abstract away less important information from the source code. Hence diagrams obtained via RE also require manual effort for simplifying them. The empirical investigation of whether this extra effort is worthwhile is still pending, but would provide us with knowledge regarding the return on investment of UML modeling in software maintenance. This topic has not been explored in detail, and this is necessary if we are to discover whether UML will have a widely acceptation in industry and how UML can gain a widely adoption in industry.

Acknowledgements. This research has been funded by the GEODAS-BC project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01). The authors would like to thank the students who have cooperated in the experiments.