Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?: A Controlled Experiment

Ana M. Fernández-Sáez
Leiden Institute of Advanced Computer Science, Leiden University
Niels Bohrweg 1, 2333 CA, Leiden, The Netherlands
+31(0)715275772
fernande@liacs.nl

Michel R.V. Chaudron
Joint Computer Science and Engineering Department, Chalmers University of Technology & University of Gothenburg, Hörselgången 11, Göteborg, Sweden
+46317721165
chaudron@chalmers.se

Marcela Genero
Instituto de Tecnologías y Sistemas de Información, University of Castilla-La Mancha
Paseo de la Universidad 4, Ciudad Real, Spain
+34926295300 Ext.3740
Marcela.Genero@uclm.es

Isabel Ramos
Departamento de Lenguajes y Sistemas Informáticos, University of Seville
Av. Reina Mercedes s/n, 41012, Seville, Spain
+34954552776
iramos@us.es

ABSTRACT

Context: UML has been the de facto standard notation for modeling object-oriented software systems since its appearance in 1997. UML diagrams are important for maintainers of a system, especially when the software was developed by a different team. These diagrams of the system are not always available, however, and are commonly recovered using Reverse Engineering (RE) techniques. When obtained through RE, UML diagrams have a high level of detail as compared to those developed in the forward design activity. **Method**: In this paper we report on a comparison of the attitude and performance of maintainers when using these two kinds of diagrams during the maintenance of source code. Our findings were obtained by carrying out a controlled experiment with 40 students of a Master's degree in Computer Science. **Results**: The results show a preference for forward design diagrams but do not display significant differences in task performance. The post-experiment survey results have led us to conclude that the subjects did not consider RE diagrams helpful; they found them difficult to understand, particularly the sequence diagrams. In the case of forward design diagrams, subjects considered sequence diagrams as useful, but they did not really employ them. **Conclusions**: Based on our findings, as regards performance of maintainers, there are no objective results which favor the use of one of these types of diagram in particular, i.e., UML diagrams which come from forwards design, on the one hand, and diagrams obtained from RE, on the other. Subjective opinions do, however, lead us to recommend the use of diagrams created during design. Nevertheless, we realize that the results should be considered as preliminary ones; further replications of this experiment are planned, using students and professionals, the aim being to obtain more conclusive results.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *Documentation,* and D.2.10 [**Software Engineering**]: Design - *Representation*

General Terms

Documentation. Design. Experimentation. Languages.

Keywords

Software Maintenance; UML Diagrams; Reverse Engineering; Controlled Experiment; Survey.

1. INTRODUCTION

The current increasing complexity of software projects [34] has led to the emergence of UML [25] as the de facto standard modeling notation. It first appeared in 1997 and has now become one of the most widely-used modeling languages in industry, as a tool with which to increase the understanding between customer and developer and to improve communication among team members [23]. Despite this, not all UML diagrams have the same complexity, layout, level of abstraction, origin, etc. [21], depending on many factors such as designers' experience, time-pressure and client conventions. Previous studies have shown that the style and rigor used in the diagrams may vary considerably throughout software projects [21], in addition to affecting the source code of the system in a different way [24].

We focus our research on the maintenance phase, because we know that this phase takes up the greater part of software development resources [10, 27]: "Maintenance typically consumes 40 percent to 80 percent of software costs. Therefore, it is probably the most important life cycle phase of software"; what is more: "60 percent of the budget is spent on software maintenance, and 60 percent of this maintenance is to enhance existing software". Forward design 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. The difference in the origin of the diagrams (i.e., forward design 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 product being maintained.

RE diagrams are easy to obtain without investing lots of developer effort. Given the ease of their generation and that they may be generated automatically at any time, maintainers can have up-to-date diagrams modeling the system when they need them. The problem these diagrams present is their very high level of detail, which may make them not very understandable. There are some issues related to the obtainment of diagrams with a high level of detail when they come from source code, after applying a reverse engineering technique:

- The level of abstraction is very low, due to the fact that every element from 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 following a specific design objective. After that, developers implement the source code following those diagrams. RE diagrams do not represent these rules, due to the fact that they are obtained from source code and these diagrams only reflect *how* the code was implemented, rather than *why*.
- These RE diagrams are platform-dependent, compared to forward design diagrams. For that reason, RE diagrams contain details about implementation patterns and frameworks used, which would not appear in forward design diagrams.
- After obtaining the RE diagrams, a cleaning and lay-outing process need to be performed, in order to adapt the diagram to the 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 can contain different levels of abstraction and detail, depending on the importance of diagram elements; this may make diagrams more understandable, and hence more effective.

All these facts lead us to pose our main research question: "Should software maintenance companies spend time updating their UML diagrams or should they rather use reverse engineered ones?" Our results might be useful for companies which are performing software maintenance and yet are unsure if they should continue updating their UML diagrams (as part of the project documentation) or if they might rather save that time, by generating RE diagrams in an automatic way. In this work we therefore analyze whether the different *Origins* of UML diagrams (RE vs. Forward Design) affect the work that must be carried out by a maintainer, in terms of the effectiveness and efficiency of the maintenance of source code. We carry out our analysis by means of a controlled experiment with students. Our aim is to find out if, in order to have an up-to-date version of diagrams, an effort should be made to maintain diagrams or not.

On the one hand, if we obtain better results with design UML diagrams we will have empirical results 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 thus avoids having to maintain the available diagrams (wherever these are available) reducing the time involved in maintenance tasks.

This paper is organized as follows. Section 2 presents the related work. Section 3 gives the description of the experiment. The results obtained in the experiment are set out in Section 4, whilst the threats to validity are summarized in Section 5. Finally, Section 6 outlines the main conclusions and future work.

2. RELATED WORK

Our work is mainly related to: (i) studies analyzing the comprehension of software systems, including those related to the comprehension of the UML diagrams, and (ii) empirical studies which focus on the comparison of the use or non-use of UML diagrams during the maintenance of software systems.

It is possible to find many papers related to the comprehension of the UML diagrams, which is directly related to the comprehension of the software system [5, 9]. For example, [22] analyzed the understandability of diagrams with different Levels of Detail (LoD) in the development phase. The results reflect a better understanding of diagrams when they have a high LoD. The authors of [11] investigate whether the comprehension of source code increases in the case of novice software engineers using abstract software diagrams produced in the early phase of the software development. Results show that there is no significant difference in the comprehension of source code achieved by the use or non-use of abstract software diagrams (although analysis diagrams are expected to have a lower LoD than design diagrams). An experiment similar to that presented in [22], but focusing solely on the maintenance phase, appears in [8]; i.e., it studies whether different LoD in UML diagrams might influence the maintenance of source code. In [8, 22] there is an assumption that the higher 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' performance, due to the fact that they understand the system they have to maintain better. The results from [8] are not conclusive, but show a slight tendency in favor of high LoD diagrams.

If we focus on studying the comprehension of UML diagrams, which is extremely relevant when performing maintenance tasks, we should highlight those studies which focus solely on maintenance tasks and the benefits of using different kinds of UML diagrams during this phase.

In [6], an experiment was performed to investigate whether the use of UML influences performance of maintenance tasks, in comparison to the use of source code only. This experiment investigated the costs of maintaining, as well as the benefits of using, UML documentation during the maintenance and evolution of a nontrivial system, with 20 professional developers used as subjects. These developers had to perform 5 maintenance tasks, consisting of adding new functionalities to an existing system; correctness, time and quality of the solution were measured. Source code, as well as 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 was to be updated, but that difference was not statistically significant. UML was, however, 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 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 [1]; this experiment obtained similar results.

In the work presented in [19], the experiment performed focuses on the comprehension of, and the difficulties involved in, maintaining the source code of object-oriented systems. UML diagrams were also presented to the subjects of the experiment, but they took as their sole focus an exploration of the participants' strategies and problems while they were conducting maintenance tasks on an object-oriented application. The results show that the major difficulties were related to understanding program logic, algorithms, discovering the impacts of changes, and the inheritance of the functionality. Based on one of the conclusions drawn from their work, the authors suggest a teaching technique by which to avoid these difficulties.

Finally, we should mention the use of UML diagrams as part of a Model Driven Engineering (MDE), out of which source code is generated automatically (and updated/maintained) through performing changes on the diagrams. The influence of this kind of approach on maintenance is studied in [15], where industrial experiences are summarized, based on the results of a questionnaire and an interviewing process. The authors concluded that use of MDE for maintenance might have positive and negative aspects at the same time. The time for stakeholders to understand each other can be reduced, thanks to the fact that it is easier for new staff to understand existing systems and the code is "self-documenting". But this time can also be increased, since the code generated may be difficult to understand. In relation to the time needed to maintain the software, their conclusions in summary form assert that this can be reduced. That is because the maintenance is done at the modeling level, and the traceability links are automatically generated. As before, however, this time can also be increased, since there is a need to keep models/code synchronized. In addition, the same work reports some percentages about the increase of the maintainability effort of a system when diagrams, UML or not, are used for different purposes (team communication, understanding of a problem, code generation, etc.).

As mentioned previously, there are several studies which deal with different points of view as regards the influence of UML on the software life cycle, but no study focuses on the differences between using forward design diagrams as opposed to RE diagrams in the maintenance phase. This fact, along with the importance that the results might have to the industry, has motivated us to perform a controlled experiment on this topic.

3. EXPERIMENT DESCRIPTION

The experiment was carried out at the University of Seville (Spain) in November 2011. In order to run and report this experiment, we followed the recommendations provided in several pieces of work [17, 18, 37]. The experiment followed the guidelines for reporting empirical research in software engineering [17] 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, including goal, context, variables, subjects, design, hypotheses, material, tasks, experiment procedure and analysis procedure.

3.1 Goal

The principal goal of this experiment was to investigate whether the *Origin* of UML diagrams influences the maintenance of source code. The GQM template for goal definition [2, 3] 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 respect to* the Origin of the UML diagrams*, in the context of* Computer Science students at the University of Seville".

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 at the design phase. In the second case, we set out to maintain a source code for which the UML diagrams are 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 due to the fact that they are also two of the most commonly used diagrams when designing a system [5, 7, 12].

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 Engineering UML diagrams, which are totally automated diagrams.
- **D**: UML diagrams obtained at the Design phase (i.e., forward design diagrams). These are totally manual designs.

RE diagrams are diagrams with a high level of detail since they represent all the elements in the source code. The D diagrams might also be considered as diagrams with a high level of detail because their class diagrams contain class names, attributes, operations and relationships, and their sequence diagrams contained lifelines, messages and parameters of messages. However, D diagrams do not represent all the elements in the source code, but those elements which are represented (based on human selection) are completely represented. D diagrams can therefore be considered as high level of detail diagrams while RE are higher level of detail 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 1), so it might be considered a small realistic system. In fact its size is almost double the LoC of other systems used in previous work which have nevertheless been considered as realistic systems, for example in [6]. The maintenance requirements were formulated

by the Master's supervisor. In the case of D 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 class diagrams of systems used in other previous work. This is caused by the use of different levels of abstraction for modeling, but their diagram size is still representative of realistic systems [13]. Note that the number of sequence diagrams in the RE group is 11 and the number of diagrams in D group is 21. Hence, the number of messages per diagram (226 messages for 21 diagrams in D group, and 191 messages for 11 diagrams in RE group) gives an indicator which suggests that RE diagrams should be considered as being larger and more complex. The RE diagrams were generated using the tool IBM Rational Software Architect, employing the default RE-functionality provided by this tool, followed by auto-lay-outing (also offered by the same tool). These experimental objects were presented in Spanish.

Table 1. Description of the system received.

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

We conducted the experiment in a classroom under controlled conditions. It was carried out with 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). 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). The students who participated in the experiment were volunteers selected for convenience (the students available in the corresponding course). Social threats caused by evaluation apprehension were avoided by not grading the students on their performance. Absenteeism was avoided by performing similar tasks to the exercises that would appear in their final exam.

The tasks to be performed did not require high levels of industrial experience, so we believed that the use of students could be considered appropriate, as suggested in literature [2, 14]. 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 [36], and there is the opportunity to test experimental design and initial hypotheses [31]. 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. Nonetheless, we also wish to test the findings with practitioners, in order to strengthen the external validity of the experiment.

3.3 Variable Selection

The independent variable (also called "main factor") is the *Origin* of diagrams, which is a nominal variable with two values (treatments): Design (D) or Reverse Engineering (RE). We also considered a further independent variable (called "co-factor" from nw on): *Ability*. We considered this co-factor in our efforts to investigate whether subjects' ability plays any role in the maintenance of source code, i.e., we discriminate between users according to the 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 such kinds of diagrams. A quantitative assessment of the participants' *Ability* was obtained by computing the final mark of the course they were taking. Those students with a final mark of below 5.7/10 (that number represents the median of the group) were classified as low *Ability* participants; those above that mark 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 marks.

The dependent variable is the maintainability. We measured this dependent variable by defining 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:

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

- **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:

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

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 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 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.

3.5 Experimental Design

We selected a between-subjects balanced design in which each treatment has an equal number of subjects [20]. 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, taking into account the suggestions provided in [37]. In an attempt to alleviate experience effects, we provided the subjects with a background questionnaire in the training session which took place before carrying out the experiment. We then assigned them to the 2 groups in a random manner (see Table 2), based on the marks obtained in the background questionnaire (blocked design by experience).

To avoid skewing the results of the tasks as a result of their being of different levels of difficulty, the tasks 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 2. Experimental design.

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

3.6 Experimental tasks

There were two kinds of maintenance tasks (Table 4) forming part of the modification questionnaires; both of these activities involve the changing of 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., it is a modification of a software product performed after delivery to keep a software product usable in a changed or changing environment [16]. 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 contained 3 tasks.
- **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*" [33]. In our case, bugs from source code had to be detected and fixed. We consequently analyzed the list of bugs reported by a professional Dutch IT development company (we will not give its name, due to terms 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 considered these tasks to be common, realistic tasks; this part of the experiment contained 2 such tasks. The subjects were provided with answer sheets for this kind of questions, to allow them to structure their responses.

These two kinds of tasks needed to be answered using some data collection forms, i.e., templates which had to be filled with pieces of code. We used these data collection forms to obtain a structured response which facilitated the correction of the results. The subjects were provided with answer sheets to allow them to structure their responses to do with the maintenance tasks. The reason for doing so is that maintaining source code on paper is not easy, due to space constraints, so the subjects were 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.). The answer sheets can be found at:
http://alarcos.esi.uclm.es/originUMLmaintenance/

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 seem not to be too complex, due to the number of LoCs which have to be changed, but the complexity of the task lies in the difficulty of detecting where change is to be performed on the source code, as well as how it should be carried out. It should also be taken into account that 5 tasks had to be completed in 2 hours, using a system that had never been seen by subjects. We limited the time of the experiment, to fit in with availability of subjects. Subjects are only required to maintain the system, i.e., they do not need to update diagrams according to their changes or to create test cases.

Table 3. 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)		

Table 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

In addition, at the end of the experiment execution the subjects were asked to fill in a post-experiment survey (see Table 3), 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 [26]. During the experiment execution, the subjects had to perform 5 maintenance tasks, in different orders, which are summarized in Table 4.

3.7 Experimental Procedure

In order to check the experimental material and the time duration, a pilot study was carried out before the execution of the experiment, with 6 PhD students from the University of Castilla-La Mancha in Spain. 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 for adapting 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 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 took place in the second session, in a classroom, where the students were supervised by the instructor of the course and one experimenter, and no communication between students was allowed. In order to carry out the experiment, the subjects first received the material needed to perform the maintenance tasks, and when they had finished they were given the post-experiment survey.

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), 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. We did not provide negative marks to incorrect answers, i.e., lines of code which do not solve the task.

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 results
2. We performed a Kolmogorov-Smirnov test [29] to determine the normality of distributions and a Levene [29] test to determine the homogeneity of variances. These analyses are useful for determining which parametric or non-parametric test it would be better to use.
3. Based on the results of the previous test, we tested the hypotheses formulated using the non-parametric Mann Whitney test [37] for the data collected in the experiment. This test was performed because the data obtained did not satisfy the restrictions of the ANOVA test [4] (we did not obtain normal distributions, there is no homogeneity of variances, and a sample is not greater than 30).
4. We analyzed the influence and the interaction of the co-factor (i.e., *Ability*) with the main factor (i.e., *Origin*). We used interaction plots [4] to study the interaction of the method with the co-factor. 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 over all the levels of the second factor. The resulting lines are parallel when there is no interaction and nonparallel when an interaction is present.
5. The data collected from the post-experiment survey was analyzed finally using bar graphs. In the cases in which we detected any pattern on data, we also tested these with a T-test [37], due to the nature of the data.

In all the statistical tests, we decided to accept a probability of 5% of committing a Type-I-Error [37].

3.9 Documentation and Communication

Issues such as documentation [30] and communication among experimenters [35] may influence the success or the failure of the experiment performance and future replications. We used laboratory packages and knowledge-sharing mechanisms to handle these issues. The material was originally written in Spanish, and the parts that would have to be understood by non-Spanish speakers were then translated into English. 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 three first 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 of the meeting by e-mail, to reach a shared common research plan. This phase was 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. We also held teleconferences to share knowledge among the research groups and to discuss the experimental procedure that the participants had to follow.

4. RESULTS

The following subsections show the results of the data analysis using SPSS [32].

Descriptive Statistics and Exploratory Analysis

Table 5 shows the descriptive statistics of the Maintainability measures (i.e., number of subjects (N), mean ($\overline{X}$), median, and standard deviation (SD)), grouped by the *Origin* of the UML diagrams.

At a glance, we can observe that when the subjects used design UML diagrams they obtained better values in both measures when comparing means. This indicates that forward design diagrams may, to some extent, improve the maintenance of the source code but that the differences are very slight.

In order to test the formulated hypotheses 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 test.

Table 5. Descriptive statistics for *MEffec* and *Meffic*.

Origin	N	*MEffec*			*MEffic*		
		$\overline{X}$	*Median*	*SD*	$\overline{X}$	*Median*	*SD*
RE	20	0.641	0.6818	0.165	0.00270	0. 00283	0.00079
D	20	**0.650**	0.6818	0.148	0.**00273**	0. 00303	0.00072

4.1 Influence of Origin of Diagram

In Table 6 and Table 7 we show the results for each measure of Mann-Whitney U tests, 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 we can reject the null hypothesis with the data obtained.

All these values were calculated using standard configuration of SPSS. 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.

As a final step, with each measure (*MEffec* and *MEffic*), we also tested the influence of the following co-factor: *Ability*.

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

Taking into account the results shown in Table 6, we cannot reject $H_{1,0}$, given that the p-value is 0.957, which is greater than 0.05., i.e., it would appear that 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 would be assuming a 0.946 (or 1-0.054) estimated probability of a Type II error in our assertions. Given the low value of the observed power we cannot obtain strong conclusions.

Table 6. Mann-Whitney test results for *MEffec*.

	MEffec			
	p-value	*op*	*es*	*R*
Origin	0.957	0.054	0.001	NO

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 (0.606 and 0.119 p-values, respectively).

Finally as regards *MEffec*, we tested whether the *Ability* of subjects influenced the results, but, as we expected, this did not happen, (the p-value obtained was 0.226). The interaction plot shown in Fig. 1a indicates that there was no interaction between *Origin* and *Ability* for *MEffec*. In this case, high ability participants achieved better scores than low ability ones, when both of them were using RE and D diagrams. The interaction plot also suggests that the results achieved with D diagrams are better than those obtained with RE diagrams, for high and low ability participants. This might be caused by the fact that RE diagrams contain too many details when compared with D diagrams. In particular, RE sequence diagrams are twice as large in terms of messages when compared to D diagrams. This could be because forward design diagrams only contain logical messages between objects, obviating messages between other kinds of objects, such as objects from Java packages, which are shown in RE diagrams. This difference between RE and D diagrams is based on their nature, owing to the fact that human based diagrams contain less technical details than RE diagrams because of human preferences.

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

We can observe (see Table 7) that there is no significant effect (p-value is 0.534, which is 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 we accepted the null hypothesis, we would be assuming a 0.949 (i.e., 1-0.051) estimated probability of a Type II error.

Table 7. Mann-Whitney test results for MEffic.

	MEffic			
	p-value	*op*	*es*	*R*
Origin	0.534	0.051	0.0003	NO

We also performed an analysis of the influence of the *Origin* on maintenance efficiency per type of maintenance, i.e. adaptive and corrective maintenance; again, the results were not significant (0.449 and 0.290 p-values, respectively).

We also tried to measure *MEffic* through 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 that 0.05 (i.e., p-value=0.725) but with a higher statistical power (i.e., op=0.5).

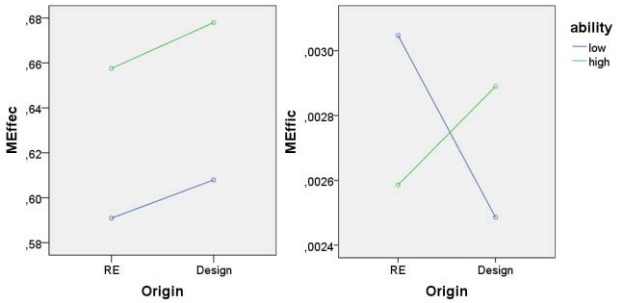

Figure 1. Interaction between Origin and Ability for a) *MEffec*; and b) *MEffic*

4.2 Influence of Ability

As with *MEffec*, the influence of *Ability* of subjects was also tested for *MEffic*, obtaining similar results (p-value=0.914), i.e., there was no statistical influence on the results of the experiment caused by the subjects' *Ability*. Once more, this was as we expected. The interaction plot shown in

Figure 1.b indicates that there was a clear interaction between *Origin* and *Ability* (both variables at the same time) for *MEffic*. In this case, high *Ability* participants achieved better scores using the D diagrams, and low *Ability* participants did better using the RE diagrams. 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. In the case of experienced maintainers, they do not need very high traceability, because using D diagrams might allow them to obtain enough information to have a correct overview of how the system works.

4.3 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 2) 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 medium difficulty (Figure 3), 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 have more experience than these Master's students, signifying that the less experienced subjects needed more time. We would also like to note that there were some subjects who did not finish the questionnaire, owing precisely to the lack of time. 10% more subjects of the RE group experienced that problem, compared to the D group.

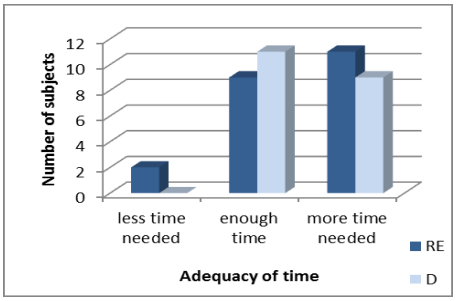

Figure 2. Subjects' answers as regards adequacy of time provided.

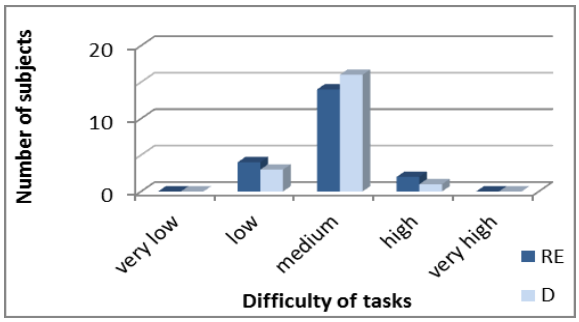

Figure 3. Subjects' answers as regards difficulty of task.

We also asked about the subjects' perception of the level of detail (LoD) of the diagrams used. The majority of the subjects who received forward design diagrams agreed with the LoD of the diagrams they received. In the case of those subjects who received RE diagrams, a greater number of subjects required less, or much less, LoD (Figure 4).

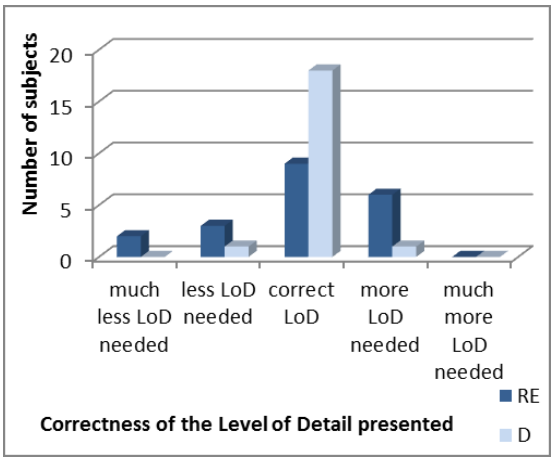

Figure 4. Subjects' answers as regards correctness of the LoD.

Subjects who received D diagrams experienced fewer difficulties when reading the diagrams used, in comparison with the RE group, as is shown in Figure 5. We tested if there was a difference as regards the difficulties experienced by subjects depending on the diagrams they used, by means of a T-test. We used that test because our sample size is less than 30 and the data follow normal distributions in this case. To carry this test out, we compared the responses of the subjects (from 1 to 5) grouped by the UML diagrams which they used (RE or D diagrams). The results of the T-test show a significant difference, because we obtained a p-value=0.001, which is lower than α=0.05. The power of the test is very high (0.957), and this therefore allows us to state that the subjects who received RE diagrams experienced more difficulties when reading diagrams than those who received forward design diagrams.

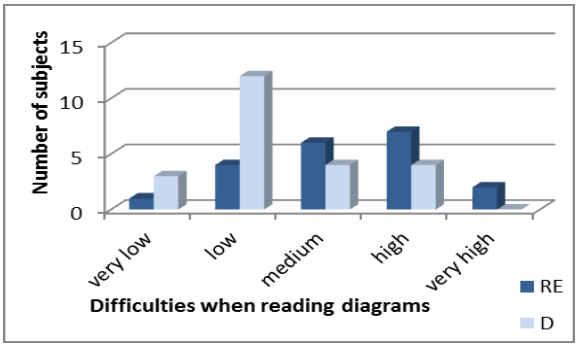

Figure 5. Subjects' answers as regards to difficulties when reading diagrams.

As part of the post-experiment survey, the subjects were required to indicate how useful the diagrams were, in general, for them as regards solving tasks. Class diagrams are considered useful in both groups, in more or less the same proportion. Having said that, however, 15 subjects of the 20 who received the RE diagrams commented that the sequence diagrams employed were not useful and were very difficult to understand, as opposed to only 6 subjects in the D group (Figure 6 and Figure 7). This

finding may have been caused by the different complexities and varying LoD in the different kinds of diagrams as explained in previous sections.

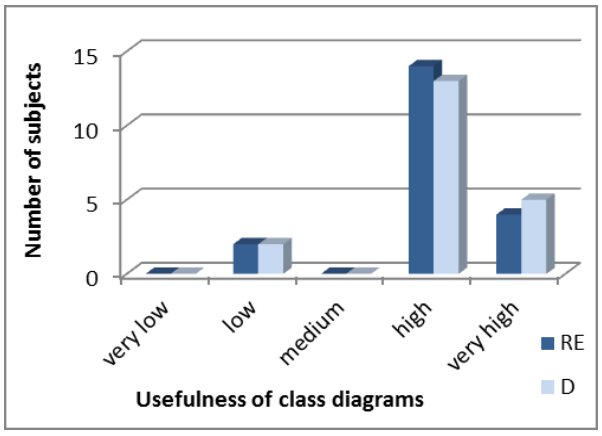

Figure 6. Subjects' answers as regards usefulness of class diagram.

After performing each maintenance task, subjects were also required to indicate which artifacts (source code, class diagrams and/or sequence diagrams) were used to solve the task. We asked subjects this in order to check if they used the diagrams to solve the maintenance tasks or not (otherwise, the measured effect would not be the influence of the different diagrams).

Source code was used by almost all subjects (i.e., 90% of subjects of the RE group, and 86% of the D group) for solving the tasks. This was expected by us, in the sense that source code is needed when it is being maintained.

After that, we analyzed if subjects used the diagrams or not. Class diagrams were also used by the majority of subjects (i.e., 80% of subjects of the RE group, and 74% of the D group). This percentage is consistent with the subjective response provided in the post-experiment survey (see Figure 6). In the case of the RE group, subjects used class diagrams in the same proportion for corrective or perfective tasks, but in the case of the D group, subjects used about 7% 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 D 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 its use was surprisingly low; in general, only 33% of subjects used it (the same percentage of use both in the RE and the D groups). That is consistent with subjects' opinion of the RE group (Figure 7), in which they indicate that they did not use sequence diagrams, and they also think that these are not useful diagrams for understanding the system during its maintenance. In the case of the D group, there is an inconsistency coming from the fact that subjects do not use sequence diagrams in most of the tasks, even though they considered them to be useful (see Figure 7). Subjects from both groups used the sequence diagram more for corrective tasks compared to perfective tasks (a difference of 20% and 27%, respectively). 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.

4.4 Summary and Discussion of the Data Analysis

Descriptive statistic results show that subjects using forward design UML diagrams obtained better values in both measures, indicating that forward design diagrams may, to some extent, improve the maintenance of the source code, but that the differences are very slight.

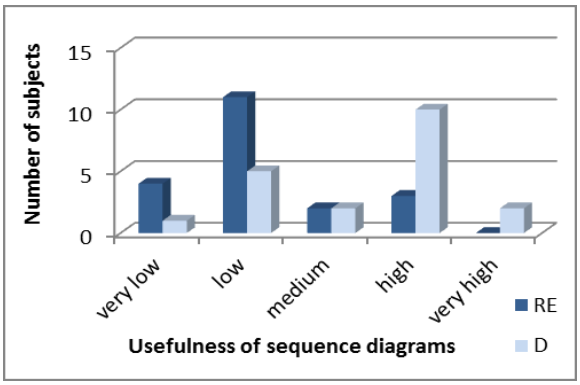

Figure 7. Subjects' answers as regards usefulness of sequence diagrams used.

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 III, 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. The results are therefore not conclusive. However, these results are considered to be preliminary, and further replications are needed.

Despite these drawbacks, we have ensured that the experimental results were not influenced by other co-factors such as the *Ability* of the subjects. If we focus on the interaction between *Origin* and *Ability*, we can say that low ability users obtain more benefits from RE diagrams than from forward design ones in terms of efficiency. That may be due to the high traceability between RE diagrams and code. In the case of high ability users, they prefer forward design diagrams.

Moreover, if we study the results of the post-experiment survey, we can see better subjective results for the forward design 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 subjects who received RE diagrams experienced more difficulties when reading the diagrams used; this is especially true with respect to sequence diagrams.

We would like to underline that UML diagrams, class diagrams at least, are used as much as source code during maintenance tasks. The sequence diagram is less widely-used, probably because of the nature of the tasks presented during the course of this experiment (a majority of perfective tasks were required compared to corrective ones). As said before, UML diagrams are not usually updated during maintenance tasks, due to time constraints on realistic environments. But the high level of use of class diagrams during this experiment leads us to recommend companies to keep these up to date, in order to help their maintainers to perform the required tasks efficiently.

5. THREATS TO VALIDITY

We must consider certain issues which may have threatened the validity of the experiment [37]:

- **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 in the relevant literature [3]. Nevertheless, it would be immensely interesting to carry out further replications of the experiment with practitioners.

  Another threat to external validity concerns the experimental material used. There are no threats related to the material used, since the UML diagrams and source code employed pertain to a real case, representative of an industrial system (business information system). 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 (due 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. It is also the case that the small number of subjects might influence the results of the experiment- This is common in empirical software engineering, however, due to the nature of the field (it is people who are required, rather than specific software or hardware).
- **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. Furthermore, all the participants found the material provided, the tasks, and the goals of the experiment to be clear, as the post-experiment survey questionnaire results showed. Another 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 participants were not allowed to communicate with each other; we prevented this happening by monitoring them during the run of the experiment. When the experiment was 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, as well as by social threats. We performed the experiment in a really short period of time, due to the subjects' constraints. The scarce amount of time allowed to the subjects for them to perform the tasks could influence the results of this experiment, as could the small number of tasks, which was due once more 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, 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.
- **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. Those 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.

6. CONCLUSIONS AND FUTURE WORK

The main concern of the research presented in this paper is the use of a controlled experiment to investigate whether the choice to use either design or reverse engineered UML diagrams influences the maintainer's performance when modifying source code. The importance of this research is based on the fact that software maintenance takes up the greater part of software projects. The use of reverse engineering techniques is an attempt to automate the generation and/or update of documentation for these kind of tasks, which could very well save time (and consequently money) in maintenance projects.

The experiment was carried out by 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. The statistical results, specifically the descriptive ones, show a very slight tendency towards getting better results when using UML diagrams obtained in the design phase; i.e. following a model-centric approach. Based on the results of the post-experiment survey, it is also important to notice that subjects preferred forward design diagrams for understanding and maintaining a system. This is true even though their performance is not so very much better with design diagrams, compared to how they do with RE diagrams. Due to the fact that software maintenance is still a human-based process in most companies, this highlighting of maintainers' perceptions, which are in favor of using forward design diagrams, is very important.

Class diagrams are important artifacts which are widely used by maintainers. However, UML diagrams are not usually updated when changes are performed on the source code. This goes against a proper use of the diagrams, a fact that obliges us to recommend companies to keep them up to date and thus help their maintainers to perform the required tasks efficiently.

It also needs to be said that significant results were obtained which show that subjects who received RE diagrams experience more difficulties when reading the diagrams used, especially the sequence diagrams. Although subjects who received design diagrams felt sequence diagrams to be highly useful, as they expressed in the post-experiment survey, only a small number of subjects actually used the diagrams. In the case of the RE diagrams group, subjects did not use them, but they also point out that they are not very useful, due to their low level of readability. Even though the experiment showed no significant difference in task performance, the subjective opinions of the participants do favor forward design diagrams.

We are conscious that these results should be considered as preliminary. Further replications of this experiment are planned, with students and professionals, in an effort to obtain more conclusive results. Nevertheless, the preferences expressed by the subjects in this first experiment, through the post-experiment

survey, give us grounds to encourage software developers, albeit with caution, to follow a model-centric approach. This implies beginning the development of a software system by building the corresponding UML diagrams, as well as keeping these up-to-date, thereby making it easier to perform maintenance tasks.

It is also important to note that we expected a better performance with design diagrams because something that requires effort (totally manual diagrams, like D diagrams) would obviously appear to be "better" than something that is totally automated; however, the results did not support this to the extent expected. According to the objective results of the experiment, there is only a slight tendency in favor of D diagrams, while according to the subjective results obtained from the post-experiment survey this tendency appears to be greater. This forces us to consider the return of the investment of UML modeling in software maintenance, which will be taken into account in future research.

The UML is widely used in the software industry [5, 28]. The results obtained are therefore useful for all those companies that exploit this notation as a support for software maintainers when performing maintenance tasks.

7. ACKNOWLEDGMENTS

This research has been funded by the following projects: MEDUSAS (CDTI-MICINN and FEDER IDI- 20090557), ORIGIN (CDTI-MICINN and FEDER IDI-2010043(1-5), and GEODAS-BC (Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01). The authors would like to thank the students who have cooperated in the performance of the experiment.

8. REFERENCES

[1] Arisholm, E., Briand, L.C., Hove, S.E. and Labiche, Y. The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation. *IEEE Transaction on Software Engineering*, *32* (6). 365-381.

[2] Basili, V., Shull, F. and Lanubile, F. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, *25* (4). 456-473.

[3] Basili, V. and Weiss, D. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, *10* (6). 728-738.

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

[5] Dobing, B. and Parsons, J. How UML is used? *Communications of the ACM*, *49* (5). 109-114.

[6] Dzidek, W.J., Arisholm, E. and Briand, L.C. A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *IEEE Transactions on Software Engineering*, *34* (3). 407-432.

[7] Erickson, J. and Siau, K. Theoretical and practical complexity of modeling methods. *Communications of the ACM*, *50* (8). 46-51.

[8] Fernández-Sáez, A.M., Genero, M. and Chaudron, M.R.V. Does the level of detail of UML models affect the maintainability of source code? *Proceedings of the 2011th international conference on Models in Software Engineering (MODELS'2011), Experiences and Empirical Studies in Software Modelling Workshop (EESSMod)*, Wellington, New Zealand, 2011, 134-148.

[9] Genero, M., Fernández-Sáez, A.M., Nelson, H.J., Poels, G. and Piattini, M. A systematic literature review on the quality of UML models. *Journal of Database Management*, *22* (3). 46-70.

[10] Glass, R. *Facts and fallacies of software engineering*. Addison-Wesley, 2002.

[11] Gravino, C., Tortora, G. and Scanniello, G. An empirical investigation on the relation between analysis models and source code comprehension *ACM Symposium on Applied Computing (SAC'2010)*, ACM, Sierre, Switzerland, 2010, 2365-2366.

[12] Grossman, M., Aronson, J. and McCarthy, R. Does UML make the grade? Insights from the software development community. *Information and Software Technology*, *47* (6). 383-397.

[13] Heijstek, W. and Chaudron, M.R.V. Empirical Investigations of Model Size, Complexity and Effort in Large Scale, Distributed Model Driven Development Processes - A Case Study *35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2009)*, Patras, Greece, 2009.

[14] Höst, M., Regnell, B. and Wohlin, C. Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, *5* (3). 201-214.

[15] Hutchinson, J., Whittle, J., Rouncefield, M. and Kristoffersen, S. Empirical Assessment of MDE in Industry *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, ACM, New York, NY, USA, 2011, 471-480

[16] ISO/IEC ISO/IEC 14764-1999: Software Engineering Maintenance.

[17] Jedlitschka, A., Ciolkowoski, M. and Pfahl, D. Reporting experiments in software engineering. in Shull, F., Singer, J. and Sjøberg, D.I.K. eds. *Guide to Advanced Empirical Software Engineering* Springer Verlag, 2008.

[18] Juristo, N. and Moreno, A. *Basics of software engineering experimentation*. Kluwer Academic Publishers, 2001.

[19] Karahasanovic, A. and Thomas, R. Difficulties experienced by students in maintaining object-oriented Systems: an empirical study *Australasian Computing Education Conference (ACE'2007)* 2007, 81–87.

[20] Kirk, R.E. *Experimental design. procedures for the behavioural sciences*. Brooks/Cole Publishing Company, 1995.

[21] Lange, C.F.J., Chaudron, M.R.V. and Muskens, J. In practice: UML software architecture and design description. *IEEE Software*, *23* (2). 40-46.

[22] Nugroho, A. Level of detail in UML models and its impact on model comprehension: A controlled experiment. *Information and Software Technology*, *51* (12). 1670-1685.

[23] Nugroho, A. and Chaudron, M.R.V. Evaluating the impact of UML modeling on software quality: An industrial case study *12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, Springer, Denver, CO, USA, 2009.

[24] Nugroho, A. and Chaudron, M.R.V. A survey into the rigor of UML use and its perceived impact on quality and productivity *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM 2008)*, ACM, New York, NY, USA, 2008, 90-99.

[25] OMG. The Unified Modeling Language. Documents associated with UML version 2.3: http://www.omg.org/spec/UML/2.3/, Object Management Group, 2010.

[26] Oppenheim, A.N. *Questionnaire design, interviewing and attitude measurement*. Pinter Publishers, 1992.

[27] Pressman, R.S. *Software engineering: a practitioners approach*. McGraw Hill, 2005.

[28] Scanniello, G., Gravino, C. and Tortora, G. Investigating the role of UML in the software modeling and maintenance - a preliminary industrial survey *12th International Conference on Enterprise Information Systems*, Funchal, Madeira, Portugal, 2010, 141–148.

[29] Sheskin, D. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman and Hall, 2007.

[30] Shull, F., Mendonça, M.G., Basili, V.R., Carver, J., Maldonaldo, J.C., Fabbri, S., Travassos, G.H. and Ferreira, M.C. Knowledge-sharing issues in experimental software engineering. *Empirical Software Engineering*, *9* (1-2). 111-137.

[31] Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N. and Rekdal, A.C. A survey of controlled experiments in software engineering. *IEEE Transaction on Software Engineering*, *31* (9). 733-753.

[32] SPSS. *SPSS 12.0, syntax reference guide*. SPSS Inc., Chicago, USA, 2003.

[33] Swanson, E.B. The dimensions of maintenance *Proceedings of the 2nd international conference on Software engineering (ICSE 1976)*, IEEE Computer Society Press, San Francisco, California, United States, 1976, 492-497.

[34] Van Vliet, H. *Software engineering: principles and practices* Wiley, 2008.

[35] Vegas, S., Juristo, N., Moreno, A., Solari, M. and Letelier, P. Analysis of the influence of communication between researchers on experiment replication *Proceedings of the ACM/IEEE international symposium on Empirical software engineering (ISESE'2006)*, 2006, 28-37.

[36] Verelst, J. The influence of abstraction on the evolvability of conceptual models of information systems *International Symposium on Empirical Software Engineering (ISESE'04)*, 2004, 17-26.

[37] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslén, A. *Experimentation in software engineering: an Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.