Empir Software Eng (2016) 21:212–259
DOI 10.1007/s10664-014-9354-4

Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments

Ana M. Fernández-Sáez · Marcela Genero · Danilo Caivano · Michel R. V. Chaudron

Published online: 24 December 2014

Abstract Although the UML is considered to be the de facto standard notation with which to model software, there is still resistance to model-based development. UML modeling is perceived to be expensive and not necessarily cost-effective. It is therefore important to collect empirical evidence concerning the conditions under which the use of UML makes a practical difference. The focus of this paper is to investigate whether and how the Level of Detail (LoD) of UML diagrams

Communicated by: Richard Paige

A. M. Fernández-Sáez · M. Genero
ALARCOS Research Group, Department of Technologies and Information Systems, University of Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain

M. Genero
e-mail: Marcela.Genero@uclm.es

D. Caivano
Department of Informatics, University of Bari, Via E. Orabona 4, 70126 Bari, Italy
e-mail: danilo.caivano@uniba.it

e-mail: danilo.caivano@searandp.com

D. Caivano
SER&Practices s.r.l, Spin Off Company of the University of Bari, Via E. Orabona 4, 70126 Bari, Italy

M. R. V. Chaudron
Joint Computer Science and Engineering Department of Chalmers, University of Technology and University of Gothenburg, SE-412 96 Gö teborg, Sweden
e-mail: chaudron@chalmers.se

e-mail: chaudron@liacs.nl

A. M. Fernández-Sáez (✉) · M. R. V. Chaudron
Leiden Institute of Advanced Computer Sciences, Leiden University, NielsBohrweg 1, 2333 CALeiden, The Netherlands
e-mail: anamaria.fernandez.saez@gmail.com

Ana M. Fernández-Sáez
e-mail: fernande@liacs.nl

impacts on the performance of maintenance tasks in a model-centric approach. A family of experiments consisting of one controlled experiment and three replications has therefore been carried out with 81 students with different abilities and levels of experience from 3 countries (The Netherlands, Spain, and Italy). The analysis of the results of the experiments indicates that there is no strong statistical evidence as to the influence of different LoDs. The analysis suggests a slight tendency toward better results when using low LoD UML diagrams, especially if used for the modification of the source code, while a high LoD would appear to be helpful in understanding the system. The participants in our study also favored low LoD diagrams because they were perceived as easier to read. Although the participants expressed a preference for low LoD diagrams, no statistically significant conclusions can be drawn from the set of experiments. One important finding attained from this family of experiments was that the participants minimized or avoided the use of UML diagrams, regardless of their LoD. This effect was probably the result of using small software systems from well-known domains as experimental materials.

Keywords UML diagrams · Software maintenance · Level of detail · Controlled experiment · Replication · Family of experiments

1 Introduction

The maintenance phase is the phase of the life cycle that absorbs a significant number of software development resources (Glass 2002; Pressman 2005): "Maintenance typically consumes 40 to 80 % of software costs. Therefore, it is probably the most important life cycle phase of software" and "60 % of the budget is spent on software maintenance, and 60 % of this maintenance is to enhance existing software". It is therefore important to attempt to improve maintainers' performance by helping them to understand the requirements and the system being maintained (especially in the case of critical systems). The comprehension of a program could consume half of the time spent by developers on maintenance (Fjeldstad and Hamlen 1979). The presence of proper documentation might thus help them in this part of the maintenance (Tryggeseth 1997). Program comprehension may be particularly benefited by graphical documentation (Arisholm et al. 2006; Dzidek et al. 2008) since, as is commonly stated, "*a picture says more than 1,000 words*". Several modeling languages have therefore emerged, which are either domain-specific modeling languages or general-purpose modeling languages. They appeared as a means to improve the customer's understanding and, in general, communication among team members (Nugroho and Chaudron 2009). The wide application of modeling has led to the development of numerous informal and formal approaches for modeling, such as Entity Relationship Diagrams (ERD) (Wieringa 2003) with which to model data, Specification and Description Language (SDL) (Glässer et al. 2003) with which to model telecommunication systems, or formal modeling languages such as Z (Spivey 1989) and B (Abrial 1996). The Unified Modeling Language, or UML (OMG 2010), is an Object Oriented modeling notation which first appeared in 1997 and has now become one of the most widely used modeling languages in industry and the de-facto software modeling notation (Grossman et al. 2005; Erickson and Siau 2007). However, there is still frequent resistance to model-based development in many software organizations since the use of the UML is perceived to be expensive and not necessarily cost-effective (Arisholm et al. 2006). This is even worse for organizations that use agile methods to develop software (Cohen et al. 2004). It is thus important to investigate whether the use of the UML can make a practical difference and justify the costs sustained. It is also important to study under which conditions (for example, in which type of software

projects, with which programming languages, type of maintainers, type of maintenance, size of system, and so on) the UML can make a practical difference.

Obviously, not all UML diagrams have the same complexity, layout, level of abstraction, etc. Furthermore, previous studies have shown that the style and rigor used in the diagrams may vary considerably according to the software project (Lange et al. 2006) and affect the source code of the system in a different way (Nugroho and Chaudron 2008). On the one hand, the different purposes for which a diagram may be intended (for example: architecting solutions, communicating design decisions, detailed specification for implementation, or automatically generating implementation code) signifies that the same system can be represented with different styles. On the other hand, 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 using a Reverse Engineering (RE) technique. RE diagrams are easy to obtain without having to invest in a lot of developer effort, as opposed to forward design diagrams which need to be manually generated and updated, and the level of detail of the RE diagrams is extremely high. Given the ease of the generation of RE diagrams and the fact that they can be generated automatically at any time, it is possible for maintainers to have up-to-date diagrams modeling the system when they need them. However, their Level of Detail might be a factor to be considered in order to ensure the effectiveness of the diagrams during software maintenance. If is not possible to reverse engineer a UML diagram, then it is necessary to create it by hand during the development phase (to avoid the need to create it during later maintenance). A question therefore arises: "To what LoD is it necessary to detail the diagrams as part of the document of a system process being used in order not to use more resources in its creation than the paybacks that might originate from it in a future maintenance based on a model-centric approach?" Our experience allows us to state that the use of incomplete documentation and of a subset of the entire software system on which a maintenance operation impacts is quite common in the software industry. A similar question to the previous one then arises: "To what LoD is it necessary to update the UML documentation in order for it to be synchronized with the source code and thus attain understandability benefits during further maintenance?

The aforementioned considerations motivated us to perform a family of experiments in order to analyze whether and how the understandability and modifiability of source code vary when using low or high LoD UML diagrams during the maintenance of a system when a model-centric approach is used. The UML diagrams were only used to support the understanding of the system and to guide its implementation but not to automatically generate the source code. We focused on a model-centric approach rather than an MDD (Model-Driven Development) approach because the LoD expected to be used with the purpose of automatically generating source code would probably always be sufficiently high for details in the source code not to be lost, but this is not the hypothesis of this paper.

We considered a specific type of maintenance in order to restrict the context of this family of experiments. We specifically selected perfective maintenance for two reasons: (i) it appears to be one of the most commonly used types of maintenance (systems are constantly evolving to become more complete and to offer more functionalities); and (ii) the requirements of this type of maintenance could be easily reproduced when using systems from well-known domains.

The family consisted of a controlled experiment and three replications carried out with students from 3 different countries (The Netherlands, Spain, and Italy). The participants were 81 subjects with different abilities and levels of experience with the UML and with the Java programming language (the experimental material is available at: http://alarcos.esi.uclm.es/maintenanceUMLfamilyExperimentsLoD/).

The main goal of this paper is to present a thorough description of this family of experiments, the main findings and the practical implications.

This paper is organized as follows. The related work is presented in Section 2. Section 3 presents the family of the experiments, while the results obtained from the experiment and the three replications are discussed in Section 4. The meta-analysis and the summary and discussion of the data analysis are presented in Sections 5 and 6, respectively. The threats to validity and the lessons learnt are highlighted in Section 7. The paper concludes in Section 8 with final remarks and future work.

2 Related Work

In (Fernández-Sáez et al. 2013b) a systematic mapping study of empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code is presented. This systematic mapping study discovered 38 papers (published from January 1997 to January 2010) containing 66 empirical studies (primary studies) related to this topic. Only the following two works are directly related to the use of UML diagrams in source code maintenance:

- In the work of Dzidek (Dzidek et al. 2008) an experiment was performed to investigate whether the use of the UML influences maintenance in comparison to the use of source code only. This experiment investigated 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 developers had to perform 5 maintenance tasks consisting of adding new functionalities to the existing system, and 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 the UML for maintainers. In terms of time, the UML subjects took longer if the UML documentation had to be updated, but that difference was not statistically significant. However, the UML was always beneficial in terms of functional correctness (introducing fewer faults into the software), because the subjects in the UML group obtained, on average, a practically and statistically significant 54 % increase in the functional correctness of changes. The UML also helped produce better quality code 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 by Arisholm (Arisholm et al. 2006) and obtains similar results. In our experiments we compare the influence of two different kinds of UML diagrams on source code maintainability rather than the presence of UML diagrams per se. Moreover, the UML diagrams in our experiments do not need be updated.
- Arisholm et al. 2006) presented the results of a controlled experiment carried out to assess the impact of UML design diagrams on software maintenance. Software professionals were involved. The authors analyzed the time taken to perform the modifications to the system, the time spent 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 had been given UML diagrams. As in (Dzidek et al. 2008), the participants' ability and experience were not analyzed as regards the comprehensibility and modifiability of source code. One difference with regard to our study is that the authors analyzed the effect of UML based

documentation (a use case diagram, sequence diagrams for each use case, and a class diagram) on modification tasks performed both on UML diagrams and source code.

The remaining primary studies in the systematic mapping study were focused on the understandability of UML diagrams. This topic is, in some respects, related to the understandability of the system since one influences the other, but these kinds of studies are not considered in this paper.

We updated the search period to August 2013 and found several additional empirical studies that were to some extent related to the content of the present paper, and these are described as follows:

- (Scanniello et al. 2013) used a family of controlled experiments to discover that the use of UML analysis 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. Analysis diagrams might be considered as low LoD diagrams while design diagrams might be considered as high LoD diagrams. The most remarkable difference between that paper and the work presented herein is that here we consider the UML diagrams produced in the design phase with different levels of detail.
- In the work of Karahasanovic (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 the 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.
- With regard to the influence of the LoD of UML diagrams in software development, Nugroho presents an empirical experiment focused on the understandability of UML diagrams with different LoDs (Nugroho 2009). The paper in question measures the correctness and efficiency of 53 computer science Master's degree students as regards comprehending UML diagrams. The results show a better understanding of diagrams when they have a high LoD. There are some differences between the work presented by Nugroho and the family of experiments summarized in this paper. On the one hand, the experiment presented by Nugroho focuses solely on the comprehension of UML diagrams and the subjects did not have the source code of the system. On the other hand, Nugroho focuses on their use in the development phase and not on maintenance.
- In a preliminary study by Scanniello (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 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 of Scanniello (Scanniello et al. 2012) because we compare low LoD diagrams with high LoD diagrams rather than with no UML diagrams, and the dependent variable is also different because Scanniello's work (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.

– A comparison of the attitude and performance of maintainers when using Forward Fesigned (FD) diagrams vs. Reverse Engineered (RE) diagrams during the maintenance of source code is shown by Fernández-Sáez et al. (2013a, 2014) through a family of controlled experiment with students (40, 51 and 78 respectively). The statistical results, show a tendency to obtain better results when using UML diagrams (concretely class diagrams), that were hand-made during the design phase. When considering the qualitative results of the post-experiment survey, it was also noteworthy that the subjects preferred FD diagrams when understanding and maintaining a system. The post-experiment survey results also led the authors to conclude that the subjects found RE diagrams, and particularly the sequence diagrams, difficult to understand.

All the related work is summarized in Table 1, thus providing the reader with an overview of the main information in order to compare the empirical studies. The columns in 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.

The analysis of the literature presented above reveals that, with regard to the usefulness of UML diagrams in helping during source code maintenance, empirical evidence is scarce (see summary in Table 1). Nonetheless, the existing evidence shows that there is, to some extent, a favorable tendency in that the use of UML diagrams benefits source code maintenance (Arisholm et al. 2006; Dzidek et al. 2008).

The completeness of the documentation, which is significantly related to the level of detail of a UML diagram, has been identified as one of the most important quality factors to affect the overall quality of the documentation (Garousi et al. 2013). The influence of the different levels of detail of UML diagrams during software development has been studied (Nugroho 2009), but there are no clear results as to their influence on software maintenance because analysis models do not influence maintenance (Scanniello et al. 2013) while design models do

Table 1 Summary of related works

Ref	Type of empirical study	Goal	Subjects	Independent variables	Dependent variables	Experiment design	Tasks	Results
(Dzidek et al. 2008)	1 experiment	To investigate whether the use of UML influences maintenance in comparison to the use of only source code.	20 professional developers.	The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code).	-Time needed to change source code. -Time needed to change source code+UML diagrams. -Functional correctness and quality of the solution.	Between-subject design.	Modification tasks in source code and in UML diagrams.	The UML subjects took more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce better quality code when the developers were not yet familiar with the system.
(Arisholm et al. 2006)	2 experiments	To investigate whether the use of UML influences maintenance in comparison to the use of only source code.	Undergraduate students (22 and 76, respectively).	The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code).	-Time needed to change source code. -Time needed to change source code+UML diagrams. -Correctness of the change. -Quality of the change.	Between-subject design.	Modification tasks in source code and in UML diagrams.	The UML subjects took more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce better quality code when the developers were not yet familiar with the system.
(Scanniello et al. 2013)	Family of experiments (1+3)	To investigate whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code.	Undergraduate students (24, 22, 22 and 18, respectively).	The use of UML analysis models (possible values: presence or absence of UML diagrams accompanying source code).	-Comprehen-sion level of the source code. -Capability of a maintainer to modify source code.	Within participants counterbalan-ced experimen-tal design.	Comprehension and modification tasks and subjective questions.	UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability.
(Nugroho 2009)	1 experiment	To determine the influence of	11 undergraduate students.	The level of detail of UML diagrams	-Understand-ability of source code.	Between-subjects balanced design.	Comprehension and modification tasks	No significant results in favor of high or low

Table 1 (continued)

Ref	Type of empirical study	Goal	Subjects	Independent variables	Dependent variables	Experiment design	Tasks	Results
		different levels of detail (LoD) of UML diagrams in source code maintenance.		(possible values: high or low LoD).	-Modifiability of source code.		and subjective questions.	LoD. There is a slight tendency in favor of low LoD diagrams.
(Karahasanovic and Thomas 2007)	1 experiment	To investigate the comprehension and the difficulties involved in maintaining object-oriented systems.	34 undergraduate students.	-	-Comprehen-sion of system. -Modification of systems.	Between-subject design.	Modification questions.	Detection of common difficulties as regards understanding when maintaining software systems.
(Scanniello et al. 2012)	1 experiment	To investigate whether the comprehension of source code increases when participants are provided with UML class and qsequence diagrams produced in the software design phase.	16 undergraduate students.	The use of sequence and class diagrams created in the design phase (possible values: presence or absence of UML diagrams accompanying source code).	-Comprehen-sion of source code.	Within-subjects.	Comprehension questions.	Participants comprehend source code significantly better when class and sequence diagrams are added together.

(Scanniello et al. 2012). What is more, the design diagrams obtained good results when compared to reverse engineered diagrams (Fernández-Sáez et al. 2014). In our case, we are of the opinion that low LoD diagrams are similar to analysis models, while high LoD diagrams are comparable to design models. The issues that are taken into account to state this similarity are explained in section 3.1. Given that analysis diagrams are more similar to low LoD diagrams and that design diagrams are more similar to high LoD diagrams, better results are expected when using high LoD diagrams.

The research presented in this paper is different from previous research because it pursues a different goal. In particular, we aim to discover whether high LoD diagrams help maintainers to better understand the system and perform the modifications that need to be made to the source code during the maintenance phase in comparison with low LoD diagrams. We shall thus contribute, to some extent, towards building up the body of knowledge regarding the evidence of the benefits of using UML.

3 The Family of Experiments

Families of experiments allow researchers to answer questions that are beyond the scope of individual experiments and permit the generalization of the findings from various studies, thus providing evidence with which to 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" (Pressman 2005) 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" (Lindsay and Ehrenberg 1993).

We carried out a family of experiments to investigate whether the different levels of detail (LoD) affect the work that must be carried out by a maintainer. The main goal of this paper is to present a family of experiments that consists of one experiment and three external replications. Figure 1 provides some information (i.e., the name of the experiment, the number of participants, and the context) about the experiments and their chronology.

In this family of experiments the participants were undergraduate/graduate students from different years of the Bachelor's or Master's Computer Science degree. The original experiment (denominated as E-UL) was carried out at the University of Leiden (The Netherlands) in March 2010. The first replication (denominated as R1-UCLM) was performed at the

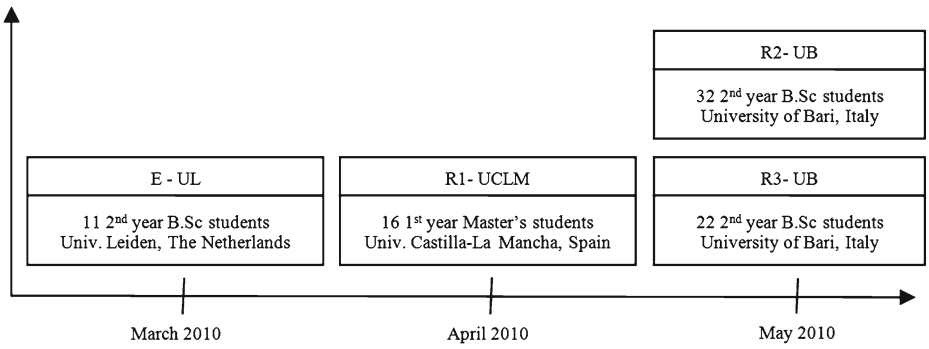

Fig. 1 Chronology of the family of experiments

University of Castilla-La Mancha (Spain) in April 2010. Two more replications (denominated as R2-UB and R3-UB, respectively) were carried out in May 2010 at the University of Bari (Italy).

The original experiment (E-UL) was performed as a pilot study with a small number of subjects, in order to obtain an initial insight into the influence of the LoD (Fernández-Sáez et al. 2012 Does the Level of Detail of UML Models Affect the Maintainability of Source Code? Proceedings of the Experiences and Empirical Studies in Software Modelling Workshop EESSMod'11 at MODELS 2011). We decided to corroborate the results obtained by carrying out a series of external replications. These experiments can be considered as strict replications of the original experiment because the essential aspects of the experimental conditions have not varied in any way: the only difference is the participants involved.

In order to run and report this family of experiments, we followed the recommendations and guidelines for conducting and reporting empirical research in software engineering provided in several works (Wohlin et al. 1999; Juristo and Moreno 2001; Jedlitschka et al. 2008; Carver 2010). For replication purposes, we created an experimental package containing the experimental material, which is available at: http://alarcos.esi.uclm.es/maintenanceUMLfamilyExperimentsLoD/

The main characteristics of each experiment and the replications are described in the following subsections, including goal, context selection, variables selection, hypothesis formulation, experimental design, experimental tasks, experimental procedure, analysis procedure and documentation and communication.

3.1 Goal

The main goal of the family of experiments, using the GQM template (Basili and Weiss 1984; Basili et al. 1999), is to: "*Analyze the level of detail in UML diagrams with the purpose of evaluating it with regard to the understandability and modifiability of source code during a model-centric development from the point of view of researchers, in the context of Computer Science undergraduate/graduate students at the following Universities: University of Leiden, University of Castilla-La Mancha and University of Bari*".

As in (Nugroho 2009), we considered that the LoD in UML diagrams should be defined as the amount of information that is used to represent a modeling element. LoD is a 'continuous' metric, but in the experiment we have taken two "extremes" - high and low LoD. Based on the LoD concept, one of our main assumptions is that the more information a diagram contains, the more is known about the concepts/knowledge described in the diagram. This knowledge would help maintainers to better understand the source code of the system and consequently modify it according to the maintenance requirements.

We decided to use 3 different types of diagrams (use case, sequence and class diagrams) since they are those most frequently used (Grossman et al. 2005; Dobing and Parsons 2006; Erickson and Siau 2007). When the LoD used in a UML diagram is low, it typically contains only a few syntactical features, such as class-name and associations, without specifying any further facts about the class. When it is high, the diagram also includes class attributes and operations, association names, association directionality, and multiplicity. In sequence diagrams, in which there is a low LoD, the messages among objects have an informal label, and when the LoD is high the label is a method name plus the parameter list. We do not consider that it is possible to distinguish between low and high LoD in use case diagrams because they are very simple diagrams, so both groups received the same set of diagrams. The elements that fit each level of detail are detailed in Table 2, and an example of the two versions of a class diagrams is shown in the Appendix.

Table 2 Levels of detail in UML diagrams

Diagram	Element	Low LoD	High LoD
Class diagram	Classes (box and name)	✓	✓
	Attributes	✗	✓
	Types in attributes	✗	✓
	Operations	✗	✓
	Parameters in operations	✗	✓
	Associations	✓	✓
	Association directionalities	✗	✓
	Association multiplicities	✗	✓
	Aggregations	✓	✓
	Compositions	✓	✓
Sequence diagram	Actors	✓	✓
	Objects	✓	✓
	Messages in informal language	✓	✗
	Messages with formal language (name of a method)	✗	✓
	Parameters in messages	✗	✓
	Labels in return messages	✗	✓
Use case diagrams	They were the same for both treatments because they do not contain enough details to distinguish between Low or High LoD		

3.2 Context Selection

The experimental material used in E-UL was in English. However, the material was translated for the replications. A native Spanish speaker translated all the material into Spanish in the case of R1-UCLM, while a native Italian speaker translated all the material into Italian in the case of R2-UB and R3-UB. The replicators supported the native speakers and helped them when needed (e.g., in the translation of technical terms).

Two experimental objects were used:

- **System A**: a library application from which a user can borrow books.
- **System B**: a sport center application from which users can rent services (tennis courts, etc.)

System A is a library extracted from (Ericksson et al. 2004). We decided to use this because it was a representative system, it was complete (source code and diagrams were available) and it gave us a starting point with which to compare our results. It was only possible to compare the results obtained from the subjects who received System A with a high LoD with the result from (Karahasanovic and Thomas 2007) because the material received was the same (although the tasks were different). The other treatments, i.e., the documents related to System A with a high LoD, were created by us for the experiment). The source code in System A contains 3,369 lines of code and the documentation consists of 5 use case diagrams with 17 use cases, 5 class diagrams with 25 classes and 15 sequence diagrams with 242 messages (Table 3).

System B is a Sport center application created as part of the Master's degree Thesis of a student from the University of Castilla-La Mancha. The source code in System B contains 5,123 lines of code and the documentation consists of 5 use case diagrams with 22 use cases, 4 class diagrams with 16 classes and 21 sequence diagrams with 226 messages (Table 3).

Table 3 Description of the systems received

	#Class diagrams	#classes	#Sequence diagrams	#messages	#Use case diagrams	#use cases	LoC
System A	5	25	15	242	5	17	3,369
System B	4	16	21	226	5	22	5,123

Both systems are desktop applications from well-known domains and are more or less the same size, based on the metrics mentioned below. Owing to their origins and size we consider them as small systems, based on to the size classification given in (Scanniello et al. 2010). The documentation of both systems can be considered to be sufficiently realistic for small-sized development projects of the following kinds: in-house software (the system is developed inside the software company for its own use) or sub-contracted (a sub-contractor develops or delivers part of a system to a main contractor) (Lauesen 2002). They can also be considered as realistic systems because they were written in Java, which is one of the most common programming languages used in companies (based in our experience). We considered well-known domains in order to avoid the extra cognitive effort required in the case of non-well-known domains, which might have biased the results.

As explained previously, the documentation of System A was extracted from a book about UML (Ericksson et al. 2004) from which the RUP (Rational Unified Process) process is taught, and it was therefore expected that the diagrams would be created by following that modeling process. System B, meanwhile, was created as part of a Master's degree Thesis by following an RUP process. In both cases, therefore, the diagrams and the source code were developed by adopting an incremental development process similar to that suggested by Bruegge and Dutoit (2010). In both cases the diagrams were created with the intention of supporting the understanding of the system and guiding the implementation of the source code within the aforementioned model-centric development processes but the intention was not related to the automatic generation of source code from it. As in the majority of developments of this kind, the diagrams were available in order to provide insights into how the system had been created and to increase the understandability in possible further maintenance. UML diagrams may therefore provide a "lens" for the code that may assist with maintenance tasks, but it is important to highlight that the UML diagrams used during a model-centric development would not faithfully represent the reality of the source code.

In order to check the material used in the experiment, one of the authors first reviewed the documentation of the two systems to find possible issues. No modifications of any note were needed to improve either the documentation (e.g., typographical errors from the models were removed) or the source code (e.g., the source code was indented). Next, and with the intention of checking the complexity and duration of the experiment, a pilot study was carried out with 3 PhD students from the University of Leiden and 2 PhD students from the University of Castilla-La Mancha. Owing to the fact that the intended participants in the family of experiments would have less experience than those who took part in in the pilot study (Master's Degree students and PhD students respectively), we decided to provide more time in the execution of the experiments than in the pilot. Based on the results of these two pilots, we considered that the two experimental objects fitted the time constraints of the experiment while being sufficiently realistic as regards the small maintenance operations that novice software maintainers perform within a software company (Cohen et al. 2004).

The experimental material delivered to the subjects consisted of the UML diagrams (use case, class and sequence diagrams) and the JAVA code of Systems A and B, the paper-based answer sheets and the post-experiment questionnaire.

We conducted all the experiments in research laboratories under controlled conditions. The subjects' knowledge was sufficient for them to understand the systems provided, and they had roughly the same background. The participants, who were grouped by experiment, had the following characteristics:

- **E-UL**: The experiment was carried out by 11 Computer Science students from the University of Leiden (The Netherlands) who were taking the *Software Engineering* course in the second-year of their B.Sc. Almost all the students were non-repeating course students (only one was repeating the course) and virtually none of them had any industrial experience (one had experience as a programmer and another as a designer). Their knowledge of UML diagrams was general and had been acquired during the Software Engineering course. Based on their responses to the preliminary test, which contained background questions, we assumed that most of the subjects considered their own UML knowledge to be medium for use case and class diagrams, and low for sequence diagrams. Moreover, most of the subjects' knowledge of JAVA language was medium/high, while all their knowledge of C++ language was medium/high, and this language is very similar to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out.
- **R1-UCLM**: The first replication was carried out by 16 Computer Science students from the University of Castilla-La Mancha (Spain) who were taking the *Quality of Information Systems* course in the second year of their Master's . More than 80 % of the students were non-repeating course students and virtually none of them had any industrial experience (three had experience as programmers and another as a maintainer). Their knowledge of UML diagrams had been acquired during other courses (such as Software Engineering I and Software Engineering II). The subjects' responses in the preliminary test again showed that most of them considered their own UML knowledge to be medium/high for use case, class and sequence diagrams. All of them had medium/high knowledge of JAVA language, and they also had knowledge of C and C++ languages which are very similar languages to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out.
- **R2-UB**: The second replication was carried out by 32 Computer Science students from the University of Bari (Italy) who were taking the *Software Engineering* course in the second year of their B.Sc. Three quarters of the students were non-repeating course students and virtually none of them had any industrial experience (two has experience as programmers). Their knowledge of UML diagrams was general and had been acquired during the Software Engineering course. The responses provided in the preliminary test led us to assume that most of the subjects considered their own UML knowledge to be medium/high for class diagrams, and medium/low for use case and sequence diagrams. Almost half of them considered their knowledge of JAVA language to be medium/high, while the rest considered it to be low. In addition, almost all of them considered their knowledge of C and C++ languages to be medium/high, and these languages are

very similar to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out.

- **R3-UB**: The third replication was carried out by 22 Computer Science students from the University of Bari (Italy) who were taking the S*oftware Engineering* course in the second year of their B.Sc. 80 % of the students were non-repeating course students and virtually none of them had any industrial experience (three had experience as programmers and another as a maintainer). Their knowledge of UML diagrams was general and had been acquired during the Software Engineering course. Most of the subjects considered their own UML knowledge to be medium/high for use case and class diagrams, and medium/low for sequence diagrams (based on the results of the preliminary test). Most of them considered their knowledge of JAVA language to be medium/high, while half of them considered their knowledge of C and C++ languages to be medium/high, and these languages are very similar to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out. The students who participated in the whole family of experiments were volunteers who were selected for convenience (the students available in the corresponding course). The subjects who participated in the experiment were not graded on their performance, and they obtained extra points in their final marks.

The participants selected were students in their last academic year because they can be considered as novice software engineers in real companies. They were required to work as individual maintainers.

3.3 Selection of Variables

The independent variable (also called the "main factor") is the LoD, which is a nominal variable with two values: low LoD and high LoD.

The dependent variables are:

- **Modifiability**, denoting the capability of the source code to be modified by a software engineer (modeler, designer, maintainer, etc.).
- **Understandability**, denoting the capability of the source code to be comprehended by a software engineer (modeler, designer, maintainer, etc.).

These two variables were selected because understandability and modifiability are considered in literature to have a direct influence on maintainability (Arisholm et al. 2006; Cruz-Lemus et al. 2010; Roehm et al. 2012; ISO/IEC 2014). The following measures were defined in order to measure these dependent variables:

- **Understandability Effectiveness (*UEffec*):** This measure reflects the ability to correctly understand the system presented. It is calculated with the following formula: *number of correct answers / number of questions*.
- **Modifiability Effectiveness (*UEffic*):** This measure reflects the ability to correctly modify the system presented. It is calculated with the following formula: *number of correctly performed modification tasks / number of modification tasks*.
- **Understandability Efficiency (*MEffec*):** This measure also reflects the ability to correctly understand the system presented. It is calculated with the following formula: *number of correctly answered questions / time spent*.

- **Modifiability Efficiency (*MEffic*):** This measure also reflects the ability to correctly modify the system presented. It is calculated with the following formula: *number of correctly performed tasks / time spent*.

These measures were applied to the responses obtained from a questionnaire composed of 3 understandability questions and 3 modifiability questions (further details of this are provided in Section 3.6) in order to obtain a quantitative evaluation of these dependent variables. A higher value of the measures reflects a better understandability/modifiability.

Additional independent variables (called "co-factors") were considered according to the experimental design selected, and their effect was controlled and analyzed:

- **Order**. We analyzed whether the order in which the LoD were used by the subjects biased the results. This was analyzed because of the design selected (see Table 4), i.e., the variation in the order of application of each method (low LoD, high LoD), which was used with the intention of alleviating learning effects.
- **System**. This factor indicates the systems (i.e., A and B) used as experimental objects. The design selected for the experiment (see Table 4) forced us to choose two application domains in order to avoid learning effects. Although our intention was that the systems used would not be a confounding factor owing to their size similarities and domains, this might not have been that case, and the system might have influenced the subjects' performances. We therefore studied the effect of the systems used on the results obtained. The following diagrams were used during the experiment:

- A-H: high LoD diagrams of system A.
- A-L: low LoD diagrams of system A.
- B-H: high LoD diagrams of system B.
- B-L: low LoD diagrams of system B.

- **Ability**. A quantitative assessment of the participants' ability was obtained by computing the average of the grades of the courses taken. The students from the University of Leiden and Castilla La Mancha with average grades below 8/10 (a 5 is needed to pass the course) were classified as low ability participants, otherwise high. In the replications conducted with students at the University of Bari, the threshold was 27/30. Students with average grades below 27/30 (an 18 is needed to pass the course) were therefore classified as low, otherwise high. We used different threshold values in the experiments conducted in The Netherlands, Spain and Italy because different grading systems are used in these countries. We considered this cofactor in our efforts to investigate whether subjects' abilities play any role in the maintenance of source code, i.e., we discriminated between users according to their respective levels 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.

Table 4 Experimental design

RUN 1	LoD			RUN 2	LoD		
		Low	High			Low	High
System	A	Group 1	Group 2	System	A	Group 3	Group 4
	B	Group 3	Group 4		B	Group 2	Group 1

3.4 Hypotheses Formulation

Based on the assumption explained above, that the more information a diagram contains, the more is known about the concepts/knowledge described in the diagram, the following 4 null hypotheses have been formulated and tested:

> **$H_{1,0}$**: There is no significant difference in the subjects' understandability effectiveness when working with UML diagrams modeled using high or low levels of detail.
> **$H_{2,0}$**: There is no significant difference in the subjects' understandability efficiency when working with UML diagrams modeled using high or low levels of detail.
> **$H_{3,0}$**: There is no significant difference in the subjects' modifiability effectiveness when working with UML diagrams modeled using high or low levels of detail.
> **$H_{4,0}$**: There is no significant difference in the subjects' modifiability efficiency when working with UML diagrams modeled using high or low levels of detail.

The goal of the statistical analysis will be to reject these null hypotheses and possibly to accept the alternative ones (e.g., Hn1 = ¬ Hn0). All the hypotheses are two sided because we did not postulate that any effect would occur as a result of the LoD.

3.5 Experimental Design

When designing the experiment we attempted to alleviate several issues that might threaten the validity of the research being carried out by considering the suggestions provided in (Wohlin et al. 1999). We selected a balanced factorial design in which the group-interaction acted as a confounding factor (Kirk 1995). This ensured that each subject worked on different experimental objects (System A or B) in two runs, using a different LoD (High or Low) each time. This design permits learning and fatigue effects to be mitigated and allows the effect of cofactors to be studied. Table 4 presents the outline of the experimental design.

All the experiments are balanced with regard to the number of participants assigned to the method (LoD). The assignment of the subjects to each group was performed by using Ability as a balancing factor, i.e., before carrying out the experiment or the replications we provided the subjects with a background questionnaire and assigned them to the 4 groups, randomly distributing experience equally throughout the groups (blocked design by experience) in an attempt to alleviate experience effects.

3.6 Experimental Tasks

We asked the subjects to perform the following three kinds of tasks (all of them written on paper and not using computers):

- **Understandability tasks**: This block of tasks consisted of answering 3 multiple choice questions concerning the semantics of the system, i.e., the semantics of diagrams and the semantics of code. These questions were multiple choice questions and were used to obtain *UEffec* and *UEffic*. An example of this kind of question is shown in Fig. 2. Each of these tasks had a score of 1 point when correct and 0 when false.
- **Modifiability tasks**: The intention of these tasks, which are also called perfective maintenance tasks, was to extend the functionality of the system and to improve the services provided (Lientz and Swanson 1980). In our case, new functionalities had to be added to the system, with the subjects receiving a list of 3 independent new requirements

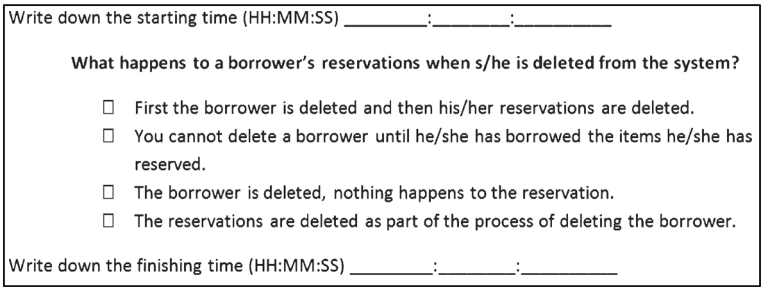

Fig. 2 Example of Understandability task

(Table 5) which had to be addressed by modifying the system code and thus adding/changing certain functionalities.

Neither of the systems used had special maintenance needs because they were not used in production, and the maintenance tasks were therefore created specifically for this family of experiments. We attempted to create a list of requirements by adding functionalities that seemed to be logical requirements (owing to the fact that they had come from well-known domains). We choose these types of maintenance tasks because they are frequent in real environments and they are also doable by individual maintainers in the time with which the participants were provided. Other types of maintenance, like corrective maintenance (i.e., activities intended to remove errors or bugs from the software), might be more frequent but could require a lot of time to detect where it is necessary to work in the case of isolated maintainers. These tasks were used to calculate the modifiability measures (*MEffec* and *MEffic*). The results of these modifiability tasks have different complexities: from adding/changing a couple of lines of code to adding the source code of a complete class. The modifiability tasks needed to be answered by using data collection forms, i.e., templates or answer sheets which had to be filled in 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 related to the maintenance tasks. The reason for doing this was that maintaining source code on paper is not easy owing to space constraints, so the subjects were required to write changes to the

Table 5 Summary of modifiability tasks

System	Task	Summary of modifiability task descriptions	Maximum mark (in points)
A	T1	The library system should store its borrowers' emails.	4
	T2	A ticket showing a borrower's loans at a specific time should be generated by the system.	5
	T3	The information about requests to buy new titles should be stored by the system.	6
B	T1	The sport center system should store its customers' telephone numbers.	4
	T2	A ticket showing a customer's reservations at a specific time should be generated by the system.	5
	T3	The information about the sport center's instructors should be stored by the system.	6

source code in a structured manner on the answer sheets (format: line-number, change type, Java code, etc.). They had to fill in a different form depending on the element that they wished to maintain (a class, an attribute, etc.). An example of one of the tasks is shown on Fig. 3.

The other tasks, along with all the material used in the experiment, are available at: http://alarcos.esi.uclm.es/maintenanceUMLfamilyExperimentsLoD/. The greatest change consisted of adding a class which would need the declaration of at least 22 new statements. In general, between 1 and 3 classes needed to be modified. The complexity of the tasks might not appear to be too complex owing to the number of necessary statements that have to be changed, but the complexity of the task lies in the difficulty involved in detecting where the change to the source code should be made, along with how it should be carried out. It should also be borne in mind that 6 tasks (3 understandability and 3 modifiability tasks) had to be completed in 2 h, using systems that the subjects had never seen before. We limited the time of the experiment to fit in with the subjects' time availability. They were only required to maintain the source code, i.e., they did not need to update diagrams according to their changes or to create test cases.

- **Post-experiment tasks**: At the end of the execution of each run, the subjects were additionally asked to fill in a post-experiment questionnaire consisting of 15 questions (see Table 6) whose goal was to obtain feedback about the subjects' perception of the experiment execution, which could be used to explain the results obtained. The answers to the questions were based on a five-point Likert scale (Oppenheim 2000) or multiple choice questions.

3.7 Experimental Procedure

In order to check the material and the duration of the experiment, a pilot study was carried out with 3 PhD students from the University of Leiden before carrying out the actual experiment (Fernández-Sáez et al. 2012 Does the Level of Detail of UML Models Affect the Maintainability of Source Code? Proceedings of the Experiences and Empirical Studies in Software Modelling Workshop EESSMod'11 at MODELS 2011). 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. A pilot experiment took place before each replica in order to check that the material had been correctly translated. The Spanish version of the material was checked by 2 PhD students at the

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

You should add functionality for also storing the system's customers' phone numbers.

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 (they cannot be accessed, showed or modified. 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 finishing time (HH:MM:SS) _____:_____:_____

Fig. 3 Example of Modifiability tasks for System B

Table 6 Post experiment questions

Id	Question/Issue	Possible answers
Q1	The difficulty of the tasks	(1–5)
Q2	The training was sufficient to be able to perform the tasks	(1–5)
Q3	The clarity of the material provided	(1–5)
Q4	The task objectives were perfectly clear to me	(1–5)
Q5	The tasks I performed were perfectly clear to me	(1–5)
Q6	I did not experience difficulty in reading the diagrams	(1–5)
Q7	I did not experience difficulty in reading the source code	(1–5)
Q8	The use of high level of detail UML diagrams helps in maintenance tasks	(1–5)
Q8.1	The use of attributes in class diagrams help you to perform maintenance tasks	(1–5)
Q8.2	The use of operations in class diagrams help you to perform maintenance tasks	(1–5)
Q8.3	The use of formal method names in messages in sequence diagrams help you to perform maintenance tasks	(1–5)
Q8.4	The use of parameters in messages in sequence diagrams help you to perform maintenance tasks	(1–5)
Q9	How much time (as a percentage) did you spend looking at the diagrams in order to perform the software maintenance?	Multiple choice question
Q10	How much time (as a percentage) did you spend source code browsing?	Multiple choice question
Q11	The time for performing the experiment was adequate	Multiple choice question

1=strongly agree; 2=agree; 3 neutral; 4=disagree; 5=strongly disagree (Q2, Q4 - Q8.4)

1=very difficult; 2=difficult; 3=medium; 4=easy; 5=very easy (Q1)

1=very clear; 2=clear; 3=correct; 4=unclear; 5=very unclear (Q3)

A=more time needed; B=less time needed; C=enough time (Q11)

A. <20 %; B. >=20 and <40 %; C. >=40 and <60 %; D. >=60 and <80 %; E. >=80 % (Q9, Q10)

University of Castilla-La Mancha, and the Italian version was checked by one member of the University of Bari's academic staff.

The experiment and replications took place in two sessions of 2 h 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. No details of the experimental hypotheses were provided in this session, 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 would be used for research purposes and treated confidentially, and that their grade on the course they were taking would not be affected by the grade obtained in the experiment. After the training session, we assigned the participants to one of the 4 groups in accordance with the marks obtained in the background questionnaire, thus obtaining balanced and homogeneous groups (see Table 4).
- Execution session: The execution of the experiment took place in the second session, in a classroom, in which the students were supervised by the course instructor (a different one depending on the replication) and one experimenter (always the same one). This second session consisted of two runs, as is required by the selected experimental design shown in

Table 4. In each run, each of the groups was given a different treatment, i.e., they first received the material for the first run, and when they had finished, they received the first post-experiment questionnaire. After filling in the post-experiment questionnaire, the material for the second run was provided. When the second run had finished, the subjects received the last post-experiment questionnaire. The participants were not allowed to interact during either each laboratory run or while passing from the first run to the second. 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.

After the execution of each experiment or replication, the data collected were placed on an excel sheet, following an answering diagram constructed before the experiment was carried out. On this sheet, the answers to the understanding tasks were graded as correct or incorrect, since they were multiple choice questions. With regard to the modifiability tasks, each task has a maximum mark (see Table 5), 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 graded negatively, i.e., lines of code which do not solve the tasks.

3.8 Analysis Procedure

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

1. Analysis of the main factor:
   - 1.1 We first carried out a descriptive study of the measures of the dependent variables, i.e., understandability and modifiability (see Section 5.1).
   - 1.2 We then analyzed the characteristics of the data to determine which parametric or non-parametric test would be most appropriate. We performed a Kolmogorov-Smirnov test (Sheskin 2007) to determine the normality of distributions and a Levene test (Sheskin 2007) to determine the homogeneity of variances.
   - 1.3 Based on the results of the previous test, we then tested the null hypotheses formulated using the non-parametric Wilcoxon test (Conover 1998) for the data collected in the experiment (see Sections 5.2.1 to 5.2.4). The use of this test was possible because, in accordance with the design of the controlled experiment, we obtained paired samples.
   - 1.4 To strengthen the results of each experiment, we decided to integrate them using a meta-analysis (see Section 5.2.5). A meta-analysis is a set of statistical techniques with which to combine the different effect sizes of the experiments in order to obtain a global effect of a factor on a dependent variable.
2. Analysis of the cofactors:

   2.1. **System**: The influence and the interaction of the System cofactor with the LoD (main factor) is analyzed in Section 5.3. There could be several concerns as regards performing this kind of analysis when adopting the within-participants counterbalanced design (Kitchenham et al. 2002). We therefore used interaction plots (Devore and Farnum 1999) to study the interaction of LoD with the cofactors. Interaction plots are simple line graphs on which the means of the dependent variables (in our case *UEffec*, *UEffic*, *MEffec* and *MEffic*, one on each graph) for each level of a factor (in our case the cofactor System)

are plotted over all the levels of another factor (in our case the LoD). The resulting lines are parallel when there is no interaction and nonparallel when an interaction is present.

2.2. **Order**: The measurement differences of the dependent variables when participants used high or low LoD were analyzed by using the method suggested by Briand et al. (Briand et al. 2005). This analysis is called Order of method (one of the cofactors), and the results are shown in Section 5.4. In particular, let:

- *Diff (low)* be the differences in the values for a dependent variable that the participants achieved when performing the tasks using a low LoD first and a high LoD second, and
- *Diff (high)* be the differences in the values for a dependent variable that the participants achieved when performing the tasks using a high LoD first and a low LoD second.

In order to verify whether *Diff (high)* was statistically greater than *Diff (low)*, we used the non-parametric Mann–Whitney test (Conover 1998) for independent samples. In our case, we expected that Diff (high) would be greater than Diff (low) for all the measures *UEffec*, *UEffic*, *MEffec* and *MEffic*. The rationale was that the participants might obtain higher scores when using a high LoD in the first run, and we therefore tested the null hypothesis H_{0Order}: *Diff (low)= Diff (high)*. In the case of rejecting this hypothesis, we verified whether *Diff (high)* was greater than *Diff (low)* for all the measures (i.e., *Diff (high)> Diff (low)*).

2.3. **Ability**: The influence of the Ability cofactor was analyzed by performing non-parametric tests owing to the nature of the data obtained (see Section 5.5).

3. The data collected from the post-experiment questionnaire with the participants' subjective perceptions was eventually analyzed using descriptive statistics and illustrated with column graphs (see Section 5.6).

In all the statistical tests performed, we decided (as is customary) to accept a 5 % probability of committing a Type-I-Error (Wohlin et al. 1999) and used SPSS (SPSS 2003) as a statistical package.

3.9 Documentation and Communication

Issues such as documentation (Juristo et al. 2013) and communication among experimenters (Vegas et al. 2006) may influence the success or the failure of replications. These issues were handled by using laboratory packages and knowledge sharing mechanisms. The material was originally written in English, and was translated into Spanish and Italian for the corresponding replications. The material included: the background questionnaire, the understanding/modification questionnaires, the answer sheets, the source code and the UML diagrams (two versions: high and low LoD) and the post-experiment questionnaire. The groups of experimenters also shared a document to provide a common background in order to communicate all the terms related to the design and analysis of the experiment.

The experimenters began with an initial face-to-face meeting at which the principal ideas of the experiments were discussed and reported in minutes. All the experimenters then exchanged the minutes of the meeting by e-mail in order to agree to a shared common research plan. This phase was relevant to sharing knowledge among the experimenters and to discussing possible issues related to the study.

The experimenters used instant messaging tools and e-mails to establish a communication channel in all the phases of the study. Teleconferences also took place in order to share

knowledge among the research groups and to discuss the experimental procedure that the participants had to follow.

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 (LoD), the analysis of the influence of cofactors and the analysis of the post-experiment questionnaire.

4.1 Descriptive Statistics and Exploratory Analysis

Tables 7, 8, 9, and 10 show the respective descriptive statistics of the *UEffec*, *UEffic*, *MEffec*, and *MEffec* measures (i.e., number of subjects (N), mean (), standard error (SE), and standard deviation (SD)), grouped by LoD.

As will be observed in Tables 7, 8, 9 and 10 (first row of each table), in half of the cases in E-UL (*MEffec* and *MEffic*), the subjects obtained, on average, better results when using low LoD UML diagrams. Only in the case of *UEffic* are the results better with high LoD diagrams. In the case of *UEffec*, there were no differences in the means. This group had the same tendency as the subjects of R1-UCLM who, on average, obtained better results in all the measures when using low LoD UML diagrams.

In the case of R2-UB1, the subjects obtained, on average, better results in both measures related to understandability (*UEffec* and *UEffic*) when using high LoD UML diagrams and, on the contrary, they obtained better results in both measures related to modifiability (*MEffec* and *MEffic*) when using low LoD diagrams.

A mixture of results also appeared in the case of R3-UB2, in which the subjects obtained, on average, better results when using low LoD UML diagrams in the *UEffic* and *MEffec* measures. In the other two variables, *UEffec* and *MEffic*, the results were better with high LoD diagrams.

These results show the following:

- *UEffec*: on average, the results show a tendency in favor of high LoD in two of the experiments and another in favor of low LoD (in the other there are no differences in favor of either low or high LoD).
- *UEffic*: The results are similar to those of the *Ueffec*, i.e., an average tendency in favor of high LoD in two of the experiments and another two in favor of low LoD.

Table 7 Descriptive statistics for *UEffec*

Exp_ID	Low LoD				High LoD			
	N	$\overline{X}$	SE	SD	N	$\overline{X}$	SE	SD
E-UL	11	0.76	0.047	0.16	11	0.76	0.065	0.22
R1-UCLM	16	0.58	0.057	0.23	16	0.56	0.050	0.20
R2-UB1	32	0.54	0.574	0.33	32	0.57	0.054	0.31
R3-UB2	22	0.38	0.063	0.29	22	0.50	0.611	0.29

Table 8 Descriptive statistics for *UEffic*

Exp_ID	Low LoD				High LoD			
	N	$\overline{X}$	SE	SD	N	$\overline{X}$	SE	SD
E-UL	11	0.0030	0.00046	0.0016	11	0.0037	0.00054	0.0018
R1-UCLM	16	0.0025	0.00032	0.0013	16	0.0023	0.00028	0.0011
R2-UB1	32	0.0018	0.00030	0.0017	31 (1outlier)	0.0020	0.00032	0.0017
R3-UB2	22	0.0016	0.00041	0.0019	22	0.0016	0.00025	0.0012

- *MEffec*: on average, the participants achieved slightly better results when employing low LoD diagrams because better mean values for low LoD were achieved in all the experiments. The difference in favor of low LoD is more evident for R2-UB1 and R3-UB2.
- *MEffec*: on average, the participants achieved slightly better results when employing low LoD diagrams. Better mean values for low LoD were achieved in all the experiments, with the exception of R3-UB2.

In summary, it can be observed that when the subjects used low LoD diagrams they obtained better values in both of the measures related to modifiability (*MEffec* and *MEffic*). This indicates that low LoD diagrams may, to some extent, improve the modification of the source code. We cannot reach any conclusion related to the understandability because the differences between both groups (high and low LoD groups) are less evident.

4.2 Influence of LoD

In order to test the formulated hypotheses ($H_{1,0}$, $H_{2,0}$, $H_{3,0}$, $H_{4,0}$) we analyzed the effect of the main factor (i.e., LoD) on the measures of the dependent variables considered (i.e., *UEffec*, *UEffic*, *MEffec* and MEffic) using the Wilcoxon test. We performed this test owing to the characteristics of the data, i.e., in most cases the data were not normal and there was no homogeneity of variances (conclusions extracted after a Kolmogorov-Smirnov test and a Levene test, respectively).

The results for each measure of the Wilcoxon test are shown in the tables (Tables 11, 12, 13, and 14) in the following subsections, in which the *#obs* column describes the number of observations, the *influence* column refers to the existence of the influence of the LoD (the p-value for the statistics test is between brackets). The *statistical observed power* and the *size of the effect* are then shown. These tables also report the number of participants that achieved

Table 9 Descriptive statistics for *MEffec*

Exp_ID	Low LoD				High LoD			
	N	$\overline{X}$	SE	SD	N	$\overline{X}$	SE	SD
E-UL	11	0.44	0.066	0.22	11	0.40	0.051	0.17
R1-UCLM	16	0.61	0.047	0.19	16	0.58	0.050	0.20
R2-UB1	31	0.47	0.050	0.28	31	0.43	0.049	0.27
R3-UB2	22	0.52	0.049	0.23	22	0.47	0.059	0.28

Table 10 Descriptive statistics for *MEffic*

Exp_ID	Low LoD				High LoD			
	N	$\overline{X}$	SE	SD	N	$\overline{X}$	SE	SD
E-UL	11	0.0049	0.00052	0.0017	11	0.0045	0.00077	0.0025
R1-UCLM	16	0.0056	0.00052	0.0021	16	0.0051	0.00047	0.0019
R2-UB1	30	0.0039	0.00055	0.0030	30	0.0038	0.00054	0.0030
R3-UB2	22	0.0033	0.00038	0.0018	22	0.0037	0.00050	0.0024

better results using a low LoD (*# of low > high*) or a high LoD *(# of low < high*). The number of participants that obtained the same results using a low or high LoD are also shown *(# of low = high*).

4.2.1 Testing Understandability Effectiveness ($H_{1,0}$)

The results in Table 11 suggest that the null hypothesis ($H_{1,0}$) cannot be rejected when using the Wilcoxon test since the p-value is greater than 0.05 in the first experiment and in all the replications. This means that there is no significant difference in *UEffec* in either group (low or high LoD diagram groups).

We decided to investigate this result in greater depth by calculating the number of subjects who achieved better values when using low or high LoD models (i.e., a low LoD value is higher than a high LoD value or vice versa, respectively). As Table 11 shows, the number of subjects who obtained the same results for both treatments (high and low LoD) is relatively high in E-UL and R1-UCLM. There were more subjects who performed better with a high LoD than with a low LoD (with the exception of R1-UCLM), but the differences in comparison to the opposite group are very small in the case of E-UL (only one subject). The results suggest that there was no statistical significant difference in the *UEffec* when the participants employed low or high LoD models. Nonetheless, the statistical powers are very low, probably because of a small effect size, so we would be assuming a 0.729 to 0.949 (i.e., 1 – statistical power) estimated probability of a Type II error in our assertions. Given the low value of the observed power we cannot obtain strong conclusions.

4.2.2 Testing Understandability Efficiency ($H_{2,0}$)

As is shown in Table 12, the p-value is greater than 0.05 in all the cases, so the null hypothesis ($H_{2,0}$) cannot be rejected. This means that there is no significant difference in the *UEffic* in either group (low or high LoD group).

Table 11 Wilcoxon tests results for *UEffec*

Dep. variable	#obs	Influence (low≠high)	Statistical power	Effect size	# of low>high	# of low=high	# of low<high
E-UL	22	No (0.680)	0.051	0.001	2/11 (18 %)	6/11 (55 %)	3/11 (27 %)
R1-UCLM	32	No (0.500)	0.058	0.003	5/16 (31 %)	7/16 (44 %)	4/16 (25 %)
R2-UB1	64	No (0.661)	0.076	0.003	9/32 (28 %)	11/32 (34 %)	12/32 (38 %)
R3-UB2	44	No (0.130)	0.271	0.043	6/22 (27 %)	5/22 (23 %)	11/22 (50 %)

Table 12 Wilcoxon tests results for *UEffic*

Dep. Variable	#obs	Influence (low≠high)	Statistical power	Effect size	# of low>high	# of low=high	# of low<high
E-UL	22	No (0.594)	0.180	0.056	6/11 (55 %)	0/11 (0 %)	5/11 (45 %)
R1-UCLM	32	No (0.776)	0.077	0.008	9/16 (56 %)	1/16 (6 %)	6/16 (38 %)
R2-UB1	64	No (0.104)	0.081	0.005	12/32 (38 %)	1/32 (3 %)	19/32 (59 %)
R3-UB2	44	No (0.639)	0.052	0.0003	9/22 (41 %)	1/22 (5 %)	12/22 (54 %)

As in the analysis of Understandability Effectiveness, a more in-depth analysis of this measure was carried out by calculating the number of subjects who achieved better values when using the low or high LoD diagrams (i.e., a low LoD value is higher than a high LoD value or vice versa, respectively). As Table 12 shows, there were isolated cases (one or no cases in each experiment) with subjects who obtained the same *UEffic* for both treatments (high and low LoD). More subjects performed better with a low LoD than with a high LoD in E-UL and R1-UCLM but the differences are very slight (1 subject and 3 subjects, respectively). In the case of R2-UB1 and R3-UB2, the results shows a slight tendency toward obtaining better results with a high LoD than with a low LoD (with a difference of 7 subjects in the case of R2-UB1).

The results suggest that there was no statistically significant difference in the *UEffic* when the participants employed low or high LoD models. Bearing in mind that the statistical powers are very low, there is an 82–94.8 % probability of wrongly assuming the equal influence of high and low level of detail.

4.2.3 Testing Modifiability Effectiveness ($H_{3,0}$)

We tested the formulated hypothesis ($H_{3,0}$) using a Wilcoxon test. Its results, which are shown in Table 13, reflect that the p-value is greater than 0.05 in the experiment and its replications, thus forcing us not to reject the hypothesis. This means that there is no significant difference in the *MEffec* in either group (low or high LoD diagram groups).

In the case of the *MEffec*, owing to the lack of significant results, we again calculated the number of subjects who performed better with low LoD diagrams than with high LoD diagrams and vice versa. The majority of subjects obtained better results with low LoD diagrams, with the exception of R3-UB2 in which the number of subjects who performed better with low LoD was the same as the subjects who performed better with high LoD. In the case of the *MEffic*, we can see a clear tendency toward obtaining better results with a low LoD,

Table 13 Wilcoxon tests results for *MEffec*

Dep. variable	#obs	Influence (low≠high)	Statistical power	Effect size	# of low>high	# of low=high	# of low<high
E-UL	22	No (0.505)	0.069	0.009	6/11 (55 %)	1/11 (9 %)	4/11 (36 %)
R1-UCLM	32	No (0.535)	0.178	0.006	11/16 (65 %)	0/16 (0 %)	5/16 (35 %)
R2-UB1	58	No (0.309)	0.088	0.006	16/29 (55 %)	0/29 (0 %)	13/29 (45 %)
R3-UB2	44	No (0.346)	0.074	0.004	11/22 (50 %)	0/22 (0 %)	11/22 (50 %)

Table 14 Wilcoxon tests results for *MEffic*

Dep. variable	#obs	Influence (low≠high)	Statistical power	Effect size	# of low>high	# of low=high	# of low<high
E-UL	22	No (0.477)	0.070	0.010	5/11 (45 %)	0/11 (0 %)	6/11 (55 %)
R1-UCLM	32	No (0.379)	0.117	0.020	9/16 (38 %)	0/16 (0 %)	7/16 (62 %)
R2-UB1	62	No (0.922)	0.092	0.006	15/31 (49 %)	0/31 (9 %)	16/31 (51 %)
R3-UB2	44	No (0.848)	0.103	0.011	12/22 (54 %)	1/22 (5 %)	9/22 (41 %)

which is the opposite of what we had expected. The results suggest that there was no statistical significant difference in the *MEffec* when the participants employed low or high LoD models. Again, it is important to note the high probabilities of assuming a lack of influence of the level of detail owing to the low statistical powers.

4.2.4 Testing Modifiability Efficiency (H4,0)

Finally, we tested the last null hypothesis formulated ($H_{4,0}$). The results, which are shown in Table 14, suggest that the null hypothesis cannot be rejected because the p-value is greater than 0.05 in the experiment and its replications. This means that, again, there is no significant difference in the *MEffic* in either group (low or high LoD diagram groups). As in the case of the other three metrics, the statistical powers for the *MEffic* are very low, and this indicates a possible high Type II error.

As Table 14 shows, no subjects obtained the same *MEffic* for both treatments (high and low LoD). There were two cases (E-UL and R2-UB1) in which more subjects performed better with a high LoD than with a low LoD, but the differences in comparison to the opposite group are very small (1 subject). In the other two cases (R1-UCLM and R3-UB2), the subjects performed better with a low LoD.

4.2.5 Integrating the Obtained Results Through Meta-Analysis

Since there are no significant results in the tests as regards the influence of the LoD, and the effect sizes are small, we therefore decided to integrate the results of the different studies using a meta-analysis. A meta-analysis is a set of statistical techniques with which to integrate the results of various studies in order to obtain a global effect of a factor on a dependent variable. In our case the factor is the LoD of UML diagrams, and we wish to know how this affects the understandability and modifiability of source code. This technique has been used for the same purpose in other families of experiments, such as those shown in (Hannay et al. 2009; Cruz-Lemus et al. 2010; Scanniello et al. 2013).

As measures may originate from different environments and may not be homogeneous, it is necessary to obtain a standardized measure of each one, and the measures used to estimate the global size effect of the factor must then be combined. For each dependent variable, we computed the mean value obtained by the participants when using a low LoD, minus the mean value they obtained with a high LoD. We used these values to compute the Hedges' g metric (Hedges and Olkin 1985; Kampenes et al. 2007), which was used as a standardized measure. 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, in which the weights are proportional to the size of the experiment:

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

where $w_i = 1/(n_i\text{-}3)$ 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 low LoD diagrams is half a standard deviation greater than the mean 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 2006). For each measure the tool produced the forest plots depicted in Figs. 4, 5, 6 and 7. 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 low LoD diagrams improves the comprehensibility and modifiability of source code. Negative values mean that a high LoD is the improving treatment.

If we focus on the results obtained for the *UEffec* variable (see Fig. 4), the total effect is in favor of using high LoD diagrams, but the global effect size obtained is not statistically significant since the p-value is not greater than 0.05. The results are similar in the case of *UEffic* (see Fig. 5). The values obtained for the Hedge's g metric (0.123 for *UEffec* and 0.070 for *UEffic*) indicate a small size for the global effect.

If we focus on the results obtained for the *MEffec* variable (see Fig. 6), the total effect is in favor of using low LoD diagrams, but the global effect size obtained is not statistically significant since the p-value is not greater than 0.05. The results are similar in the case of *MEffic* (see Fig. 7). The values obtained for the Hedge's g metric (−0.162 for *MEffec* and −0.082 for *MEffic*) indicate a small size for the global effect.

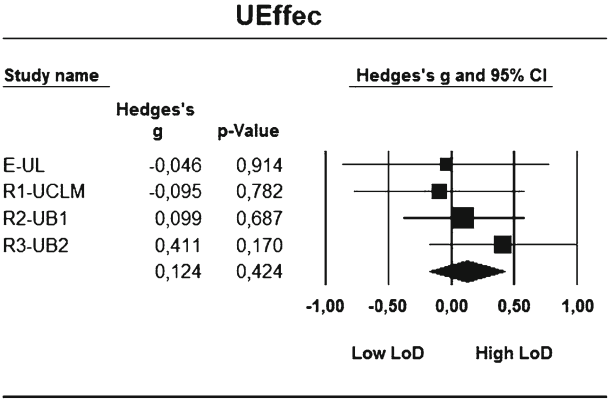

Fig. 4 Meta-analysis of *UEffec*

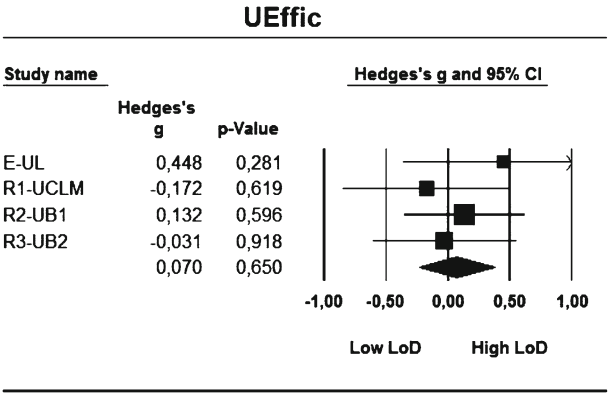

Fig. 5 Meta-analysis of *UEffic*

We can see a slight tendency toward understanding tasks in favor of high LoD diagrams while low LoD diagrams are better for modification tasks. Despite this tendency, the global effect size it is not statistically significant in any of the cases.

4.3 Influence of System

An analysis of the interaction plots shown in Figs. 8, 9, 10 and 11 shows that interaction is not present when the lines of the diagram are more or less parallel, which occurs in some cases and is the ideal situation.

In the case of E-UL (first plot of each figure), if we focus on those diagrams that have crossed lines we can separate them into two groups. The first is related to the understandability (concretely, the *UEffec* diagram), and in this case the participants appear to achieve better results when using System B with a low LoD and System A with a high LoD. These results allow us to suggest that System A has sufficient information in the low LoD version to understand the system, and when more information is introduced it appears to become more complex from the point of view of understanding. The second is related to modifiability (concretely, the *MEffec* and *MEffic* diagrams). The opposite of the understandability group

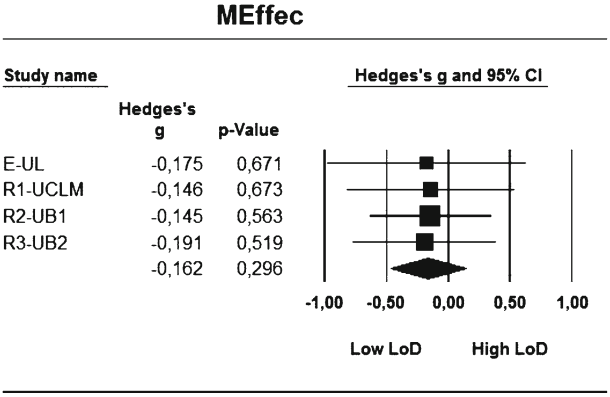

Fig. 6 Meta-analysis of *MEffec*

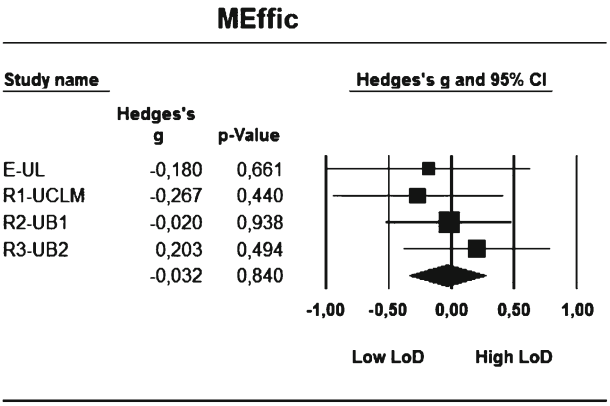

Fig. 7 Meta-analysis of *MEffic*

occurs when using System B, and the participants appear to obtain better results when using high LoD diagrams. The result seems to get worse if we add details to the diagrams when using System A. These results allow us to suggest that System B may be more complex than System A.

In the case of the results of R1-UCLM (second plot of each figure), there is less system interaction because the lines of the diagram are more or less parallel, and this is the ideal situation. Only in the case of *MEffic* do the subjects appear to perform better when adding details to System B.

The analysis of the interaction plots of R2-UB1 (third plot of each figure) shows similar results in the case of the understandablity to those obtained in E-UL (the participants seem to achieve better results when using System B with a low LoD and System A with a high LoD). But in the case of the modifiability, the results are the opposite of those obtained in E-UL (the participants seem to obtain better results when they are using high LoD diagrams).

In the last replication, i.e., R3-UB2 (fourth plot of each figure), there is not so much system interaction because in almost all cases the lines of the diagram are more or less parallel, and this is the ideal situation. Only in the case of *MEffic* do the subjects seem to perform better when adding details to System B. This pattern was also discovered in the results of R1-UCLM.

Upon focusing on measurements, the plots can be separated in two groups. The first is related to the understandability (Figs. 8 and 9), and in these cases the participants seems to achieve better results when using System B. The second is related to the modifiability (Figs. 10 and 11), and in these cases the participants seems to achieve better results when using System A.

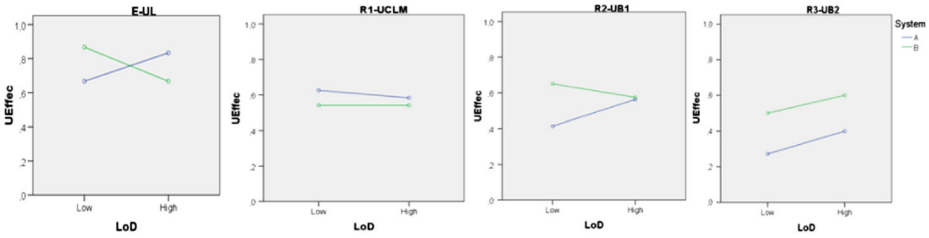

Fig. 8 Interaction between LoD and System for *UEffec*

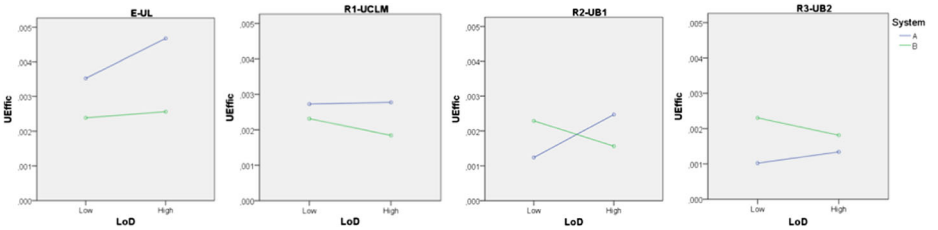

Fig. 9 Interaction between LoD and System for *UEffic*

These results allow us to suggest that the low LoD version of System A contains sufficient information to understand the system, and when more information is introduced it seems to become more complex from the point of view of the understandability. The opposite occurs when using System B. But if we focus on modifiability tasks, the participants appear to obtain better results when using high LoD diagrams and this result seems to get worse as details are added to the diagrams. The opposite occurs when using System A. These results allow us to suggest that System B seems to be more complex than System A.

4.4 Influence of Order

In this subsection we analyze the hypothesis related to the order in which the material was presented to the subjects, in which the ideal result is to accept the null hypothesis:

> *H_0: The order in which the material was presented to the subjects has no influence as regards the measure. H_1: $\neg H_0$.*

These hypotheses were tested by performing a Mann–Whitney test (see Table 15).

The results of the test revealed that the null hypothesis could not be rejected for all the experiments in the case of *UEffec*. This result means that the participants in the second run did not obtain significantly greater or smaller differences when using a low or high LoD in *UEffec*.

In the case of *UEffic*, The results of the Mann–Whitney test revealed that H_0 could not be rejected for E-UL, R2-UB1, and R3-UB2, i.e., the second run of these experiments did not obtain significantly greater differences in *UEffic*. With regard to R1-UCLM, the test revealed that the null hypothesis could be rejected: the participants in the second runs obtained significantly smaller differences in *UEffec*.

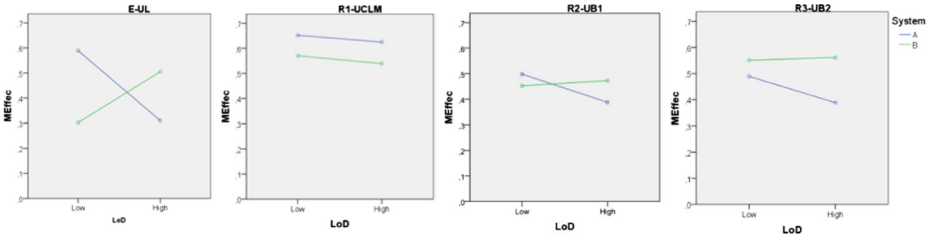

Fig. 10 Interaction between LoD and System for *MEffec*

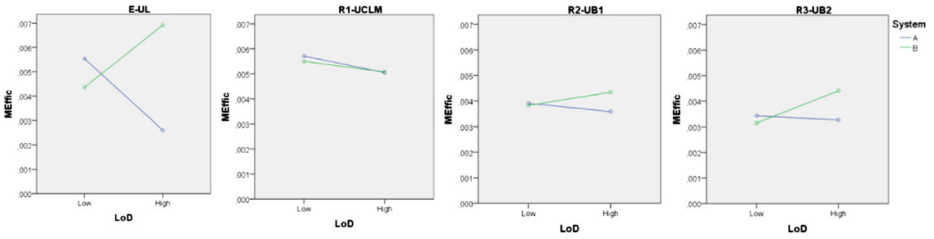

Fig. 11 Interaction between LoD and System for *MEffic*

If we focus on the measures related to modifiability, *MEffec* and *MEffic*, the results of the Mann–Whitney test revealed that H_0 could not be rejected for E-UL, R1-UCLM, and R3-UB2, i.e., the second run of these experiments did not obtain significantly greater differences in *MEffec* or *MEffic*. With regard to R2-UB1, the test revealed that the null hypothesis could be rejected since the participants in the second runs obtained significantly smaller differences in *MEffec* or *MEffic*.

4.5 Influence of Ability

In this subsection we analyze the hypothesis related to the subject's ability, in which the ideal result is to accept the null hypothesis:

H_0: The subjects' abilities have no influence on the measure. H_1: $\neg H_0$.

These hypotheses were tested by performing Mann–Whitney tests (see Table 16).

If we focus on the results of the test performed for E-UL, they signify that the null hypothesis could not be rejected for half of the variables, and the subjects' abilities thus influence the results of the experiment for half of the measures. This was the case of *UEffic* and *MEffic*, with which the test revealed that the null hypothesis could be rejected because the participants with higher abilities understood the system faster and performed better modifications. In the case of the other two measures (*UEffec* and *MEffic*) ability had no influence on their results.

In the remaining cases (R1-UCLM, R2-UB1 and R3-UB2), the results of the test indicate that the null hypothesis could not be rejected for any of the variables. The subjects' abilities did not therefore influence the results of the experiment as we had expected.

4.6 Post- Experiment Questionnaire Results

With regard to the analysis of the post-questionnaire, it is important to bear in mind that the subjects responded to 2 post-questionnaires each (one after each run). The absolute numbers in

Table 15 U-Mann Whitney tests results for the influence of order

Order	*UEffec*	*UEffic*	*MEffec*	*MEffic*
E-UL	1	0.105	0.223	0.341
R1-UCLM	0.166	0.044	0.691	0.258
R2-UB1	0.227	0.478	0.016	0.039
R3-UB2	0.98	0.216	0.188	0.46

Table 16 U-Mann Whitney tests for the influence of ability

Ability	*UEffec*	*UEffic*	*MEffec*	*MEffic*
E-UL	0.632	0.031	0.007	0.092
R1-UCLM	0.084	0.876	0.183	0.139
R2-UB1	0.161	0.085	0.899	0.829
R3-UB2	0.673	0.888	0.279	0.197

this section might therefore be higher than the total number of subjects. In order to avoid confusion, we shall refer to the number of "cases" rather than the number of "subjects" when providing these kinds of numbers.

The analysis of the answers to the post-experiment questionnaire revealed that the time needed to carry out the comprehension and modification tasks was considered to be inappropriate (more time was needed), and that the subjects considered the tasks to be quite difficult (Fig. 12), independently of the treatment received. The need for more time to perform the tasks and the consideration of the task as difficult 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 a higher ability and/or more experience than these Master's students, signifying that the less experienced subjects needed more time.

We also asked them if they had had any problems when reading the diagrams made available to them. In this case, if a subject responded "completely agree" or "agree", this indicates that s/he did not have any problems when reading the diagrams, while a response of "disagree" or "completely disagree" indicates that they did, and "neither" is a neutral response. As Fig. 13 shows, most of the subjects did not have any problems when reading the diagrams (90 vs. 38) and more particularly, the low LoD diagrams caused less problems than the high LoD diagrams (17 vs. 21). This can be explained because the structural complexity of a software diagram affects its cognitive complexity (Cant et al. 1995). Cognitive complexity refers to the mental burden that people (e.g. analysts, designers, developers, testers, maintainers, etc.) experience when building, validating, verifying or using diagrams. According to Systems Theory, the complexity of a system is based on the number of different types of elements and on the number of different types of (dynamically changing) relationships

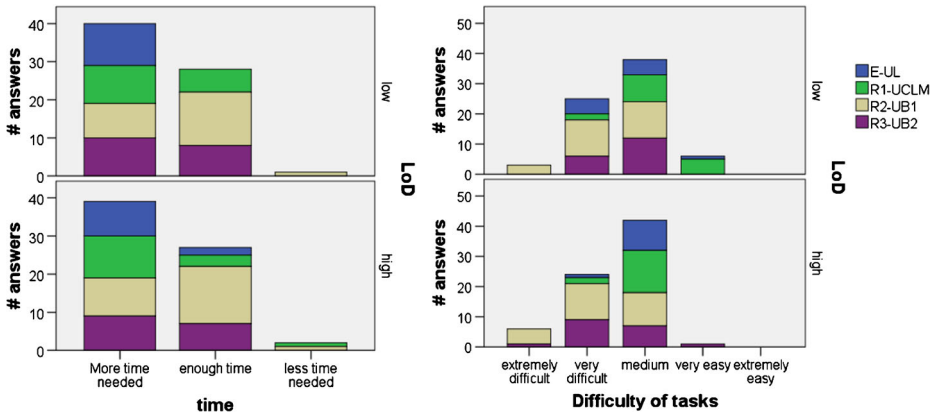

Fig. 12 Subjects' perception of the experiment

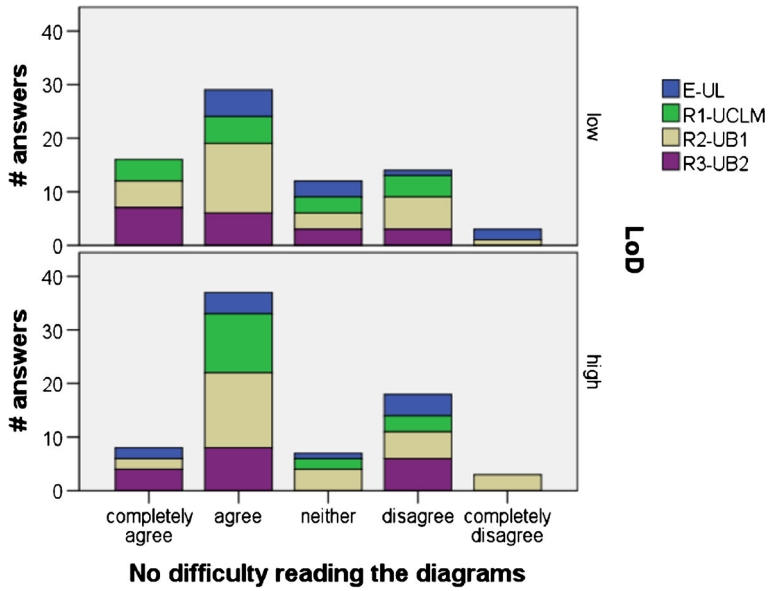

Fig. 13 Subjects' difficulties when reading diagrams

between them (Pippenger 1978). The structural complexity of a software diagram is thus determined by the elements of which it is composed, signifying that a low LoD diagram has a lower structural complexity than a high LoD diagram, and that its cognitive complexity is therefore also lower.

Moreover, some subjects (11) had difficulties when reading the source code (Fig. 14), more or less in the same proportion for the low and high LoD group in the case of E-UL. A majority of subjects also had difficulties when reading the source code (25 vs. 20) in the case of R2-UB1, and most of them were subjects from the high LoD diagrams group (14 vs. 11). In the rest of the replications, the majority of the subjects did not have any difficulties when reading the source code. A major proportion of subjects from the low LoD diagrams group in the case of R1-UCLM, and subjects from the high LoD diagrams group in the case of R3-UB2 had difficulties when reading the diagrams.

Finally, we also asked about the subjects' perceptions of some of the items that appeared in the high LoD diagrams but did not appear in the low LoD diagrams. Figure 15 shows that high LoD elements seem to be appreciated by the subjects. With regard to the histograms in Fig. 15, if a subject responds "completely agree" or "agree", this indicates that s/he thinks that the element in the question was helpful, while a response of "disagree" or "completely disagree" indicates that the elements in the question are not helpful ("neither" is a neutral response).

If we focus on the elements related to class diagrams (upper histograms) we can see that attributes are helpful in 100 cases (versus 9 cases in which the subjects do not believe them to be helpful). The same is true for the operations (102 cases vs. 4 cases). If we focus on the elements related to sequence diagrams (lower histograms) we can see that formal messages are more helpful (107) than natural language messages (2), and the same can also be said for the appearance of parameters in messages (102 vs. 5). In all these cases the graphs show that the subjects who received the high LoD diagrams agree more with the helpfulness of those elements.

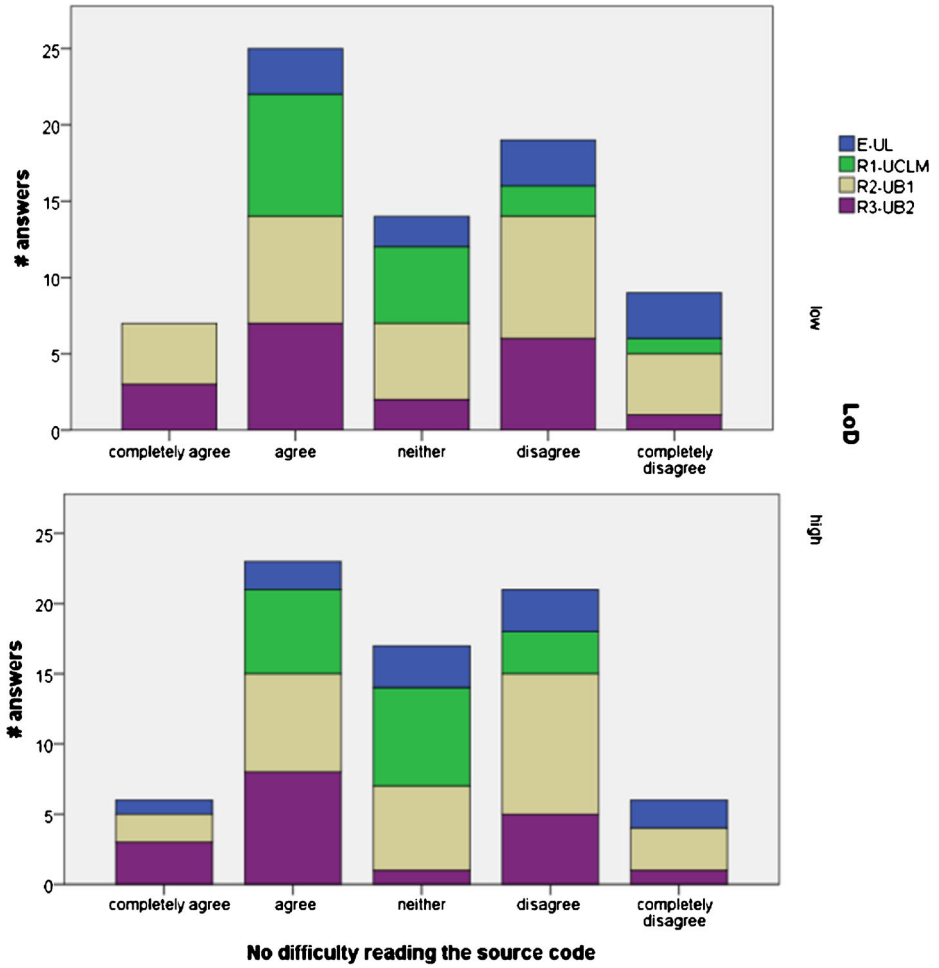

Fig. 14 Subjects' difficulties when reading source code

5 Summary and Discussion of the Data Analysis

The main findings of the family of experiments presented in this paper, which are also illustrated in Table 17, are summarized below. We shall also discuss and attempt to find an explanation for the results obtained.

The first experiment (E-UL) was performed with 11 students from the University of Leiden (The Netherlands). In this experiment we did not obtain conclusive results in favor of any of the treatments (low or high LoD). In general, the cofactors did not influence the results of the experiment. But the subjects' Ability did influence some results (*UEffic* and *MEffec*), and the System seemed to slightly influence the results of the measures (*UEffec*, *UEffic*, *MEffec* and *MEffic*). The descriptive statistics showed a tendency in favor of using low LoD, contrary to our expectations since we believed that more details in a diagram would help maintainers to perform their daily tasks. Indeed, in the results of the post-experiment questionnaire the

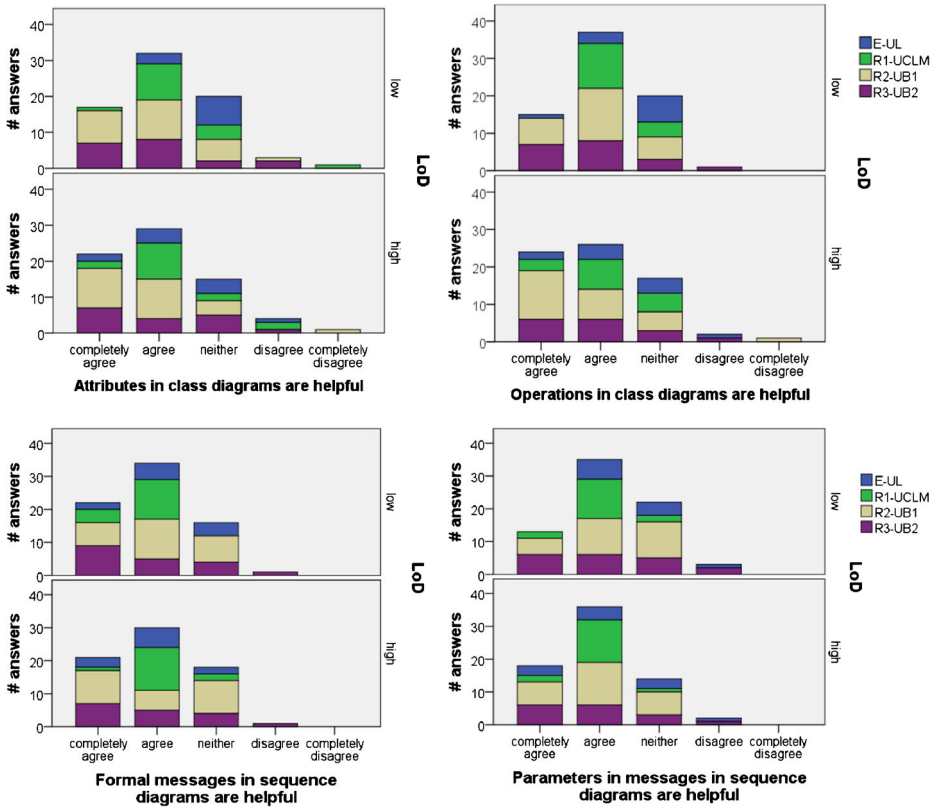

Fig. 15 Subjects' opinion of LoD

subjects with high LoD diagrams had problems when reading the diagrams. We then attempted to replicate the experiment in order to corroborate the tendency of this preliminary conclusion.

The first replication (R1-UCLM) was performed by a different group of students, concretely 16 students from the University of Castilla-La Mancha (Spain). We attempted to solve the influence of the system detected in the previous experiment by clarifying the statements of the assignments, since the E-UL subjects' answers seemed to follow an incorrect pattern which might have been caused by the statements rather than by the system itself. In this case, the influence of this cofactor was almost nil, i.e., we cancelled the influence of the system as we expected, and Order influenced only one of our four measures (*UEffic*). Following the results obtained in the first experiment, the descriptive statistics of the first replication were in favor of a low LoD, but again we did not obtain conclusive results after performing the statistical test. However, contrary to the results of the E-UL, the responses to the post-experiment questionnaire of R1-UCLM show that the subjects had more problems when reading the low LoD diagrams, and we therefore decided to perform two more replications.

In these last two replications (R2-UB1 and R3-UB2) we attempted to solve the problem of obtaining inconclusive results by increasing the number of subjects involved in the experiment, since this might help us to detect patterns in the responses and perform a more powerful statistical test. In this case, 32 and 22 students, respectively, from the University of Bari (Italy) were involved as subjects. In the second replication (R2-UB1) there were clear influences of

Table 17 Summary of results of the family of experiments

Exp_ID	Descriptive statistics (in favour of…)				Influence of LoD				Influence of System				Influence of Order			
	UEffec	*UEffic*	*MEffec*	*MEffic*	*UEffec*	*UEffic*	*MEffec*	*MEffic*	*UEffec*	*UEffic*	*MEffec*	*MEffic*	*UEffec*	*UEffic*	*MEffec*	*MEffic*
E-UL	-	H	L	L	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗
R1-UCLM	L	L	L	L	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
R2-UB1	H	H	L	L	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✓	✓
R3-UB2	H	L	L	H	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗

Descriptive statistics: "L"=better results when using LOWrather than high LoD "M"=better results when using HIGHrather than low LoD "-"=no differences when using low or high LoD

Influence of LoD

✗=hypothesis not rejected→There is no significant difference in the results when working with UML diagrams modeled using high or low LoD

✓=hypothesis rejected→There isa significant difference in the results when working with UML diagrams modeled using high or low LoD (expected value)

Influence of System

✗=hypothesis not rejected→There is no significant difference in the results when working with system A or B (expected value)

✓=hypothesis rejected→There isa significant difference in the results when working with system A or B

Influence of Order

✗=hypothesis not rejected→There is no significant difference in the results when receiving system A first and system B second, or vice versa (expected value)

✓=hypothesis rejected→There isa significant difference in the results when receiving system A first and system B second, or vice versa

the system and order, which might have been caused by learning effects or indirect effects. Attempts were made to fix these possible misunderstandings in the third replication (R3-UB2), and the cofactors did not have any influence on the results. Again, the statistical test did not provide conclusive results about the influence of the different LoDs on the understanding and modifiability of a system. Surprising, and contrary to the results of the first experiment and the first replication, the descriptive statistics of these two replications were in favor of a high LoD. Contrary to these results, the subjects had more difficulties when reading the high LoD diagrams compared to the low LoD diagrams.

After the individual analysis, a meta-analysis was performed in order to integrate the results. Its results show a tendency in favor of high LoD diagrams for both measures related to the understandability of the system (*UEffec* and *UEffic*) and in favor of low LoD diagrams for both measures related to the modifiability of source code (*MEffec* and *MEffic*). But in both cases, the results are not clearly evident owing to the values of the p-values (always lower than 0.05). These results are aligned with those obtained in the descriptive statistics.

Although the results do not seem to be in favor of a specific LoD in each experiment or replication, we noticed that in the majority of the cases, the descriptive statistics were in favor of using high LoD diagrams to understand the system, and a low LoD to maintain it.

Indeed, we performed replications of the experiment to explain or corroborate the results obtained, but we were quite frustrated by the fact that results seemed to be incoherent and random, without a clear tendency. With the goal of finding and explanation for the results obtained we decided to obtain feedback from the students who had performed the experiments, and we also wished to know the point of view of practitioners.

One year after the execution of the experiments, the majority of the students who had been involved in their realization took another course with the same professors. Occasionally, during lessons, the students asked their professors about the results obtained from them in the experiment, what hypothesis had been tested, etc.

In order to try to explain the results of the experiment, and in order to teach the students about designing and analyzing experiments, a focus-group-session explaining these concepts was planned at each location. The professors explained the details of the design of the experiment to their students (which could not been explained before in order to not influence their performance), along with the results. The results obtained (no convincing effect) were also presented to the students. They attempted to provide some feedback about the possible reasons, and in all locations they agreed that, in some cases, they had not used the diagrams, and this is therefore a clear factor that influenced the lack of results. They argued that they had not used the UML because the systems came from a well-known domain, and because the tasks were not very complex - even though it was challenging to complete them in the time available. It could be a risk to base our explanations on a focus group that came into existence 1 year after the execution of the experiment because the subjects might have forgotten the main details, but the majority of the students agreed on their responses, and we therefore considered them to be valid. It is also important to highlight that some of the students involved in the experiments during that year occasionally asked their professors about the results of the experiment, so they were obviously interested in the topic and were therefore less prone to forget it.

Another interesting opportunity for discussing and attempting to understand the results obtained was at a meeting with some practitioners from SER & Practices (a small Italian company focused on software development and software engineering research). In SER & Practices a weekly team meeting is held between the Director of the company (Danilo Caivano, one of the authors) and the three area managers (Innovation, Production and Services, Infrastructure). At the end of the meeting the Director asked the managers for help

in interpreting the results obtained as part of a second focus group. To fully understand the manager's point of view it is opportune to point out that they use a SCRUM based process for software systems development and maintenance.

A summary of the main findings of a long discussion and brainstorming session is shown below:

- The systems were too small to effectively require the use of UML diagrams or other types of technical documentation.
- The application domains of the two systems are too simple and well known to require the use of very detailed diagrams for system understanding, especially for students because these systems are usually used in textbooks or for didactical purposes.
- The maintenance request was simple and well-focused and did not therefore imply a time consuming impact analysis or the need to study the analysis or design diagrams.
- Under the conditions described above, the source code is the best documentation and the greater the level of detail in the diagram, the more difficult it is to understand. This might be caused by the fact that the time spent studying diagrams may be greater than that required to understand the system (or that part of the system was directly impacted by the maintenance request) directly through code.

Furthermore, the managers explained the way in which they use the diagrams, according to certain rules. These rules helped us to corroborate our findings, and are the following:

- When the systems maintained are large and/or technically complex (i.e., they are the result of the integration of several components developed by using different programming languages or development frameworks, etc.) they usually use the high level design diagrams/documentation in order to roughly understand the system architecture and thus which components/subsystem they have to understand and modify. They then attempt to understand the selected components directly from source code with the help of the diagrams if necessary.
- When the systems maintained are complex and difficult to understand (they gave the example of real time or embedded systems in which there are typically several functions/methods/class that are not traceable to end user functions) they prefer to use diagrams with a higher level of detail or the analysis diagrams/documentation in order to understand the system behavior. This signifies that, depending on the technical complexity/size of the system, the subjects did or did not use the design diagrams.

Finally we obtained a conclusion that we would not have been able to extract from the results of the experiment and which was obtained from occasional feedback from the 2 focus groups (students and practitioners): during the experiment it is possible that not all the subjects used the UML diagrams provided. We did not obtain the same observations from the subjects because they did not use the UML diagrams in the same way, and we were consequently unable to evaluate the LoD effects. The lack of use of the UML diagrams might have been for several reasons:

- Size of system and knowledge of the domain: The systems used were small systems and they came from well-known domains. In this case, the subjects did not need to understand the system because they already more or less knew what it was like. They only needed to navigate the source code to discover how it was implemented.

- Time constraints: We attempted to design a realistic experiment using actual systems. But the time for the experiment was very limited owing to the subjects' availability, so the tasks needed to be sufficiently simple to be carried out during the time of the experiment. The low complexity of the tasks might have meant that the subjects did not need a detailed understanding of the systems.

It might appear that the experiment was incorrectly designed. Nevertheless, we performed a pilot study to test both the material and the behavior of possible subjects, the results of which were satisfactory and helped us to improve the design of the experiment. However, the people who performed the pilot study were skilled PhD students who had more experience and abilities as regards maintaining a system by following a methodological process, particularly a model-based process.

The fact that the subjects did not use the UML diagrams provided might reflect the current situation of industrial maintainers: documentation is not taken into account when under pressure and when there is a time constraint. What is more, when the domain is well-known, or the maintainers are already familiarized with it, they minimize the time taken to read the UML diagrams and they may work directly on the source code. These results agree with those obtained in (Scanniello et al. 2013).

6 Implications of the Study

One of the main implications of the results obtained in this family of experiments is that practitioners do not need to pay so much attention to the LoD of the diagrams used during maintenance tasks when the system under maintenance comes from a well-known domain. This can only be generalized in the context of maintenance projects for Java systems performed by novice software maintainers. From the researcher perspective, the results of this family of experiments could help them to design other experiments, or complete families, taking into account the problems dealt with, such as how to deal with randomized result without a clear tendency and how to obtain an explanation for them.

We adopted a perspective-based approach (Nugroho and Chaudron 2008) to judge the implications of our family of experiments. In particular, we based our discussion on the practitioner/consultant (simply practitioner in the following) and researcher perspectives. To this end, we took advantage of the checklists proposed in (Anda et al. 2006) in order to plan and report this family of experiments. The main findings of our study and their research and practical implications are summarized as follows:

- The use of low LoD diagrams is better in comparison to that of high LoD diagrams. This statement is also supported by Briand et al.'s framework (Briand et al. 2001), which hypothesizes that high cognitive complexity (i.e., high LoD diagrams) will result in reduced understandability which impedes the analyzability, adaptability and flexibility of the diagram. This hypothesized relationship between structural complexity and external quality properties like understandability and modifiability has been repeatedly demonstrated (Siau 1999; Poels and Dedene 2000). This result is relevant from both the practitioner and the researcher perspectives. From the practitioner perspective, this result is relevant because it is useless to provide maintainers with additional information, coinciding with the results found in (Gravino et al. 2010). From the researcher perspective, it would be interesting to investigate whether variations in the context lead to different results and why these diagrams are not as useful as expected. With regard to these points, our work provides interesting insights resulting from an empirical

investigation based on 81 participants. Although this might not perhaps be surprising, this study provides the bases for future investigations that may be better focused on how the UML supports software engineers in the maintenance phase.

- The use of UML diagrams does not increase the performance of maintenance operations in small systems when developers are familiar with the domain, and they might distract the participants while performing comprehension and modification tasks. This result is relevant for the researcher because it would be interesting to investigate why participants' comprehension of source code (independently of their experience) does not improve comprehension when it is supplemented with UML diagrams (low or high LoD). A plausible justification for this result is that the traceability of the UML diagrams and the source code is reduced each time that the source code is updated without synchronization with the diagrams: the names of the entity may be different in the diagrams and source code, or the relationships between the diagrams and the source code may be changed and become more intricate in the source code than in the diagrams.
- The participants' familiarity with the problem domain of a software system might affect the comprehensibility and the modifiability of the source code, both when using and not using UML analysis models. This result could be of interest to both the researcher and the practitioner.
- When performing maintenance tasks, the source of information that the participants found more useful was the code. The participants thus perceived that UML diagrams could not benefit the execution of the tasks assigned. This result is relevant for the researcher since it might be interesting to investigate the motivation that guides a software engineer as regards trusting a source of information and how s/he exploits it to accomplish a maintenance task.
- The UML diagrams selected in the two software systems were realistic for small-sized and well-known projects. Although we are not sure that the results achieved scale to real projects, they may be of interest to practitioners working on cases in which the documentation is incomplete (e.g., in lean development processes) and the maintenance operation is executed on a subset of the source code of the entire system.
- The UML is widely used in the software industry (Cohen et al. 2004; Dobing and Parsons 2006). The results obtained are therefore useful for all those companies that exploit this notation as support for software maintainers/developers when executing maintenance operations.
- From a methodological perspective there are the following lessons: Firstly, this study benefitted greatly from introducing the post-experiment questionnaire in the replications of the experiment. This questionnaire was targeted toward clarifying some of the questions that arose after performing the first experiment. Secondly, the study showed that the subjects and conditions of a study trial should be very carefully chosen. In our case, the Ph. D. students' behavior in the trial was clearly different from that of the M.Sc. students in the actual experiment.

7 Threats to Validity

It is necessary to consider certain issues that may have threatened the validity of the experiment:

- **External validity:** External validity may be threatened when experiments are performed with students, and the representativeness of the subjects in comparison to software professionals may be doubtful. In spite of this, the tasks to be performed did not require high levels of industrial experience, so we believed that this experiment could be considered appropriate,

as suggested in literature (Basili et al. 1999; Höst et al. 2000; Bruegge and Dutoit 2010; Carver 2010). Working with students also implies a set of advantages, such as the fact that the students' prior knowledge is fairly homogeneous, there is the possible availability of a large number of participants (Verelst 2004) and there is the chance to test experimental design and initial hypotheses (Sjøberg et al. 2005). An additional advantage of using students in experiments concerning understandability and modifiability is that the cognitive complexity of the objects under study is not hidden by the participants' experience. As suggested in (Singer and Vinson 2002), ethical issues were dealt with carefully. There was no information in the raw data that could allow a particular individual to be identified. The names of the students who participated in the experiments were annotated to provide them with extra points in their courses when needed, but it was not possible to link their names to their responses. Informed consent and confidentiality are not required in these cases. The students' participation might also have been biased because they were able to benefit from it (extra points, examples of exam exercises), but their performance was not biased since all the participants obtained the same benefits, independently of their responses.

- The participants' lack of familiarity with the problem domain of a software system might affect the understandability and the modifiability of the source code, thus biasing the results since it might be an extra cognitive effort. For this reason, and owing to the time constraints, we decided to use well-known domains as part of the experiment. There are no threats related to the material used since the systems used were real, if small, and based on well-known domains. We attempted to perform an experiment by simulating real conditions, and the subjects were therefore provided with all the documentation of the system together, as a maintainer would receive all the documentation of the maintenance project together.
- **Internal validity**: The threats to the internal validity have been mitigated by the design of the experiment. Each group of subjects worked on the same systems in different orders. Nevertheless, there is still the risk that the subjects might have learned how to improve their performances from one session to the next. In all the experiments, the scores achieved by the participants were not significantly better in the second run (with the exception of R1-UCLM with *UEffic* and R2-UB1 with *MEffec* and *MEffic*). For each experiment, the internal validity threat was also mitigated owing to the fact that the participants had similar experience with the UML, software system modeling, and computer programming. Furthermore, all the participants found the material provided, the tasks, and the goals of the experiment clear, as the post-experiment survey questionnaire results showed. Another issue concerns the exchange of information among the participants. The participants were not allowed to communicate with each other. We prevented this by monitoring them both during the runs and during the break between the first and the second task. Moreover, the instrumentation was tested in a pilot study in order to check its validity. In addition, mortality threats were mitigated by offering the subjects extra points in their final marks. Another internal threat could be the influence of the language used in the documentation. English was used in the experiment that took place in The Netherlands, and this may have represented a threat owing to the fact that English is not the participants' native language. Students at Dutch Universities are, however, obliged to pass a high level English exam in order to gain entrance to the university, and we do not therefore consider this to be a high threat. The experimental material used in E-UL was in English, but the supervisors at the experiment supported the native speakers and helped them when needed (e.g., in the translation of technical terms). In the replications, the subjects were given the documentation in their native languages, signifying that this threat was mitigated. A further threat might have been the translation process; however, the experimental material was translated by native Spanish/Italian speakers.

- **Conclusion validity**: Conclusion validity concerns the data collection, the reliability of the measurement, and the validity of the statistical tests. Statistical tests were used to reject the null hypotheses. We have explicitly mentioned and discussed cases in which non-significant differences were present. Conclusion validity might also be affected by the number of observations. Further replications on larger datasets are therefore required to confirm or contradict the results.
- **Construct validity**: Construct validity may be influenced by: 1) the measures used to obtain a quantitative evaluation of the subjects' performance, 2) the comprehension questionnaires, 3) the maintenance tasks, and 4) the post-experiment questionnaire. We used a well-known and widely used measure to obtain a quantitative evaluation of comprehensibility and modifiability. The understanding/modification tasks were formulated to condition the subjects' answers in favor of neither low nor High LoD. The measures used were selected to achieve a balance between the correctness and completeness of the answers. The questionnaires were defined to obtain sufficiently complex questions without them being too obvious, and they were formulated in a similar way. The post-experiment questionnaire 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. Other possible threats to construct validity could be related to: the translation of the experimental material and social threats. The first kind of threat was reduced through the involvement of a native speaker to translate all the material used. Social threats (i.e., evaluation apprehension) were avoided since we did not grade the students on the results obtained in the experiments. As the subjects were provided with all the documentation of the system together, we were not able to control whether or not they used the diagrams. This problem could be solved by dividing the experiment into two different phases, the first of which would be related to the understanding of the system in which only the UML diagrams would be provided, and the second of which would be related to the modifiability of the system using the UML diagrams provided plus the source code. This hypothetical design would make it possible to control the factor related to the use or non-use of the diagrams, but the experiment would be less realistic because in real scenarios maintainers can access all the information when they need it.

8 Conclusions and Future Work

Many software projects do not produce complete or detailed documentation owing to time constraints. Many maintenance projects are therefore performed by maintainers who have to understand a software system based on the source code and the existing UML diagrams. An alternative might be to use UML diagrams obtained by using a reverse engineering process starting from the source code that is available. In this latter case, the diagrams would be much more detailed and less abstract than is the case of forward designed diagrams.

The Level of Detail (LoD) presented in UML diagrams might therefore be an influential factor in software maintenance tasks, but it could be different depending on the development approach used. For example in a model-based approach, the optimal LoD is not clear, but in an MDD approach a higher LoD would appear to be better as regards generating a more complete source code. We thus decided to carry out a family of four experiments in order to investigate whether the use of low LoD UML diagrams supports novice software engineers when comprehending and modifying the source code of small systems in comparison to high LoD

UML diagrams during model-centric maintenance. We have particularly focused on the perfective maintenance tasks carried out by individual maintainers.

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

The descriptive statistics might lead us to assume that high LoD diagrams are more helpful when understanding a system in comparison to low LoD, while low LoD diagrams are more helpful when carrying out maintenance tasks. Although the results obtained in this family of experiments do not appear to have a clear tendency in favor of high or low LoD based on the statistical test performed, two focus groups allowed us to extract the conclusion that UML diagrams should not be used, independently of their LoD, in the context of novice software maintainers when maintaining the source code of small realistic system from well-known domains.

The empirical evidence obtained from the family of experiments should be considered valid in the context of undergraduate/graduate students (considered as novice software maintainers) who are maintaining relatively simple systems related to well-known domains. The questionable utility of the UML in this context might be caused by the kind of systems modeled: maintainers do not need so much information about the system when performing maintenance tasks on small systems from well-known domains. In the case of novice maintainers modifying small and well-known domains, and considering the findings obtained, we would therefore recommend that companies follow a model-centric approach in order to improve the understanding of the system and source code but not invest too much in the maintenance of the documentation related to the UML diagrams alone.

Possible future directions for our current research are: (i) performing further studies considering realistic software systems related to larger and unknown domains to verify whether the findings obtained are still valid; (ii) replicating the study with practitioners; (iii) analyzing the effect of different UML notations (iv) studying the effect of different LoDs when performing other types of maintenance (for example, corrective maintenance); and (v) analyzing the effect of high or low LoD UML diagrams in projects using other kinds of development methodologies (for example MDD).

As the research presented in this work is part of a long term research effort concerning the benefits of the UML in software maintenance, we have complemented the current study with a survey completed by practitioners in order to investigate the kinds of software systems maintained by companies, their characteristics (complexity, size, domains, etc.), whether or not (in industry) companies use UML diagrams during source code maintenance, and the benefits of using the UML that practitioners perceive (https://es.surveymonkey.com/s/software-maintenance). We plan to complete the survey analysis in the near future.

Acknowledgments This work has been funded by the following projects: GEODAS-BC (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01) and IMPACTUM (Consejería de Educación, Ciencia y Cultura de la Junta de Comunidades de Castilla La Mancha, y Fondo Europeo de Desarrollo Regional FEDER, PEII11-0330-4414).

Appendix

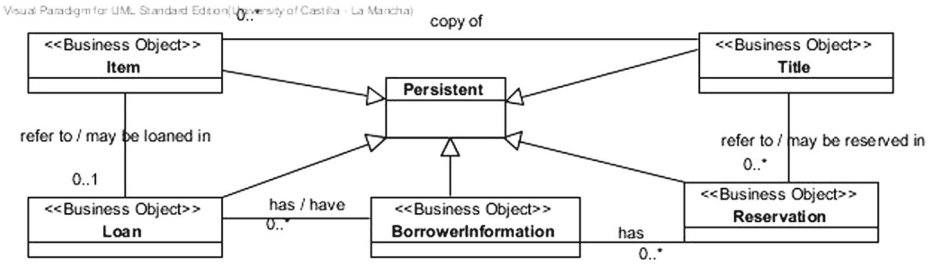

Fig. 16 Example of a low LoD class diagram

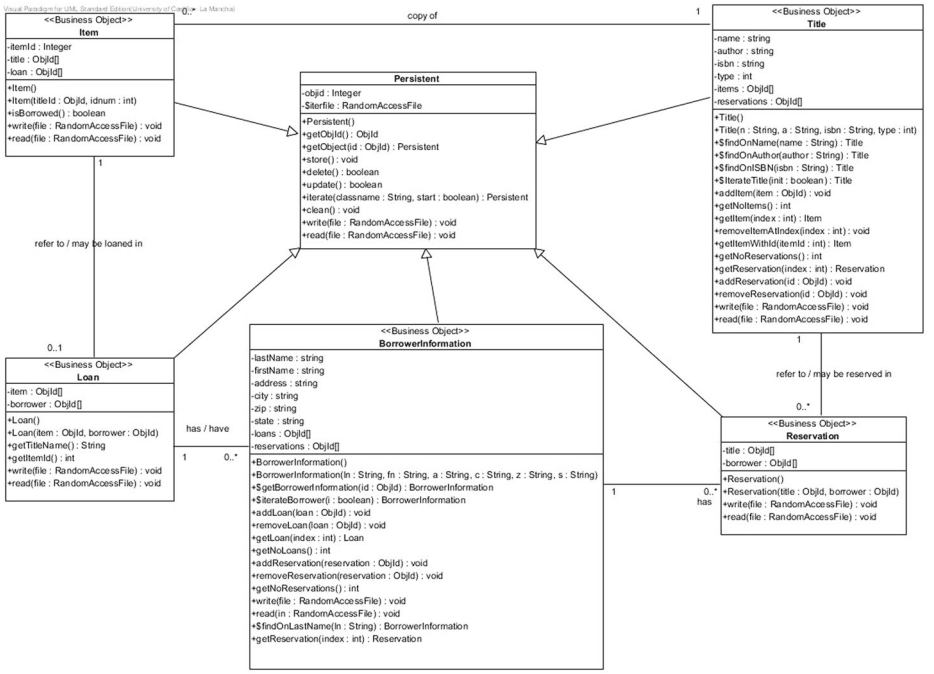

Fig. 17 Example of a high LoD class diagram

References

Abrial JR (1996) The B-Book. Cambridge University Press, New York

Anda B, Hansen K, Gullesen I, Thorsen HK (2006) Experiences from introducing UML-based development in a large safety-critical project. Empir Softw Eng 11:555–581

Arisholm E, Briand LC, Hove SE, Labiche Y (2006) The impact of UML documentation on software maintenance: an experimental evaluation. IEEE Trans Softw Eng 32:365–381

Basili V, Shull F, Lanubile F (1999) Building knowledge through families of experiments. IEEE Trans Softw Eng 25:456–473

Basili V, Weiss D (1984) A methodology for collecting valid software engineering data. IEEE Trans Softw Eng 10:728–738

Biostat (2006) Comprehensive Meta-Analysis v2. Biostat, Englewood
Briand LC, Labiche Y, Di Penta M, Yan-Bondoc H (2005) An experimental investigation of formality in UML-based development. IEEE Trans Softw Eng 31:833–849
Briand LC, Wüst J, Lounis H (2001) Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs. Empir Softw Eng 6:11–58. doi:10.1023/A:1009815306478
Bruegge B, Dutoit AH (2010) Object-oriented software engineering: using UML, patterns, and Java. Prentice Hall, Boston
Cant S, Jeffery D, Henderson-Sellers B (1995) A conceptual model of cognitive complexity of elements of the programming process. Inf Softw Technol 37:351–362. doi:10.1016/0950-5849(95)91491-H
Carver J (2010) Towards Reporting Guidelines for Experimental Replications: A Proposal. Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research (RESER) [Held during ICSE 2010]
Cohen D, Lindvall M, Costa P (2004) An introduction to agile methods. Adv Comput 62:2–67
Conover WJ (1998) Practical Nonparametric Statistics, 3rd ed. Wiley
Cruz-Lemus JA, Genero M, Caivano D et al (2010) Assessing the influence of stereotypes on the comprehension of UML sequence diagrams: A family of experiments. Inf Softw Technol 53:1391–1403
Devore JL, Farnum N (1999) Applied Statistics for Engineers and Scientists. Duxbury Press, NY
Dobing B, Parsons J (2006) How UML is used? Commun ACM 49:109–113
Dzidek WJ, Arisholm E, Briand LC (2008) A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Trans Softw Eng 34:407–432
Erickson J, Siau K (2007) Theoretical and practical complexity of modeling methods. Commun ACM 50:46–51
Eriksson HE, Penker M, Lyons B, Fado D (2004) UML 2 Toolkit. Wiley
Fernández-Sáez AM, Chaudron MRV, Genero M, Ramos I (2013a) Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?: a controlled experiment. Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering. ACM, New York, pp 60–71
Fernández-Sáez AM, Genero M, Chaudron MRV (2013b) Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. Inf Softw Technol 55: 1119–1142
Fernández-Sáez AM, Genero M, Chaudron MRV et al (2014) Are Forward Designed or Reverse-Engineered UML diagrams more helpful for code maintenance?: A family of experiments. Inf Softw Technol. doi:10.1016/j.infsof.2014.05.014
Fernández-Sáez AM, Genero M, Chaudron MRV (2012) Does the Level of Detail of UML Models Affect the Maintainability of Source Code? Proceedings of the Experiences and Empirical Studies in Software Modelling Workshop (EESSMod'11) at MODELS 2011. LNCS 7167, Wellington, New Zealand, pp 133–147
Fjeldstad RK, Hamlen WT (1979) Application Program Maintenance Study: Report to Our Respondents. Proceedings of GUIDE 48
Garousi G, Garousi V, Moussavi M, et al. (2013) Evaluating Usage and Quality of Technical Software Documentation: An Empirical Study. Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE'2013). ACM, pp 24–35
Glass R (2002) Facts and fallacies of software engineering. Addison-Wesley
Glässer U, Gotzhein R, Prinz A (2003) The formal semantics of SDL-2000: Status and perspectives. Comput Netw 42:343–358. doi:10.1016/S1389-1286(03)00247-0
Gravino C, Tortora G, Scanniello G (2010) An empirical investigation on the relation between analysis models and source code comprehension. Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'2010). ACM, pp 2365–2366
Grossman M, Aronson JE, McCarthy RV (2005) Does UML make the grade? Insights from the software development community. Inf Softw Technol 47:383–397
Hannay JE, Dybå T, Arisholm E, Sjøberg DIK (2009) The effectiveness of pair programming: A meta-analysis. Inf Softw Technol 51:1110–1122
Hedges LV, Olkin I (1985) Statistical Methods for Meta-Analysis. Academia Press, New York
Höst M, Regnell B, Wholin C (2000) Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. Proceedings of the 4th Conference on Empirical Assessment and Evaluation in Software Engineering. pp 201–214
ISO/IEC (2014) ISO/IEC 25001: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Planning and management. International Organization for Standarization
Jedlitschka A, Ciolkowoski M, Pfahl D (2008) Reporting experiments in software engineering. Guide to Advanced Empirical Software Engineering
Juristo N, Moreno A (2001) Basics of software engineering experimentation. Kluwer Academic Publishers, Boston
Juristo N, Vegas S, Solari M et al (2013) A process for managing interaction between experimenters to get useful similar replications. Inf Softw Technol 55:215–225

Kampenes V, Dybå T, Hannay JE, Sjoberg DIK (2007) A Systematic Review of Effect Size in Software Engineering Experiments. Inf Softw Technol 49:1073–1086
Karahasanovic A, Thomas R (2007) Difficulties experienced by students in maintaining object-oriented Systems: an empirical study. Proceedings of the Australasian Computing Education Conference (ACE'2007). pp 81–87
Kirk RE (1995) Experimental design. procedures for the behavioural sciences. Brooks/Cole Publishing Company, Belmont
Kitchenham BA, Pfleeger S, Hoaglin DC et al (2002) Preliminary Guidelines for Empirical Research in Software Engineering. IEEE Trans Softw Eng 28:721–734
Lange CFJ, Chaudron MRV, Muskens J (2006) In practice: UML software architecture and design description. IEEE Softw 23:40–46
Lauesen S (2002) Software Requirements: Styles and Techniques. Addison-Wesley, UK
Lientz BP, Swanson EB (1980) Software Maintenance Management. Addison -Wesley, Massachusetts
Lindsay RM, Ehrenberg A (1993) The design of replicated studies. Am Stat 47:217–228
Nugroho A (2009) Level of detail in UML models and its impact on model comprehension: A controlled experiment. Inf Softw Technol 51:1670–1685
Nugroho A, Chaudron MRV (2009) Evaluating the impact of UML modeling on software quality: An industrial case study. LNCS 5795:181–195
Nugroho A, Chaudron MRV (2008) 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, pp 90–99
OMG (2010) The Unified Modeling Language. Documents associated with UML version 2.3. http://www.omg.org/spec/UML/2.3.
Oppenheim AN (2000) Questionnaire Design, Interviewing and Attitude Measurement, 0002-New ed. Bloomsbury Academic, United Kingdom
Pippenger N (1978) Complexity Theory. Scientific American 238:
Poels G, Dedene G (2000) Measures for Assessing Dynamic Complexity Aspects of Object-oriented Conceptual Schemes. Proceedings of the 19th International Conference on Conceptual Modeling. Springer, Berlin, pp 499–512
Pressman RS (2005) Software engineering: a practitioners approach, seventh. McGraw Hill, New York
Roehm T, Tiarks R, Koschke R, Maalej W (2012) How Do Professional Developers Comprehend Software? Proceedings of the 34th International Conference on Software Engineering. IEEE Press, Piscataway, pp 255–265
Scanniello G, Gravino C, Genero M, et al. (2013) On the Impact of UML Analysis Models on Source Code Comprehensibility and Modifiability. ACM Transactions On Software Engineering And Methodology (In press) 26
Scanniello G, Gravino C, Tortora G (2012) Does the Combined use of Class and Sequence Diagrams Improve the Source Code Comprehension? Results from a Controlled Experiment. Proceedings of the Experiences and Empirical Studies in Software Modelling Workshop (EESSMoD´2012)
Scanniello G, Gravino C, Tortora G (2010) Investigating the Role of UML in the Software Modeling and Maintenance - A Preliminary Industrial Survey. Proceedings of the 12th International Conference on Enterprise Information Systems (ICEIS'2010). Funchal, Madeira, Portugal, pp 141–148
Sheskin D (2007) Handbook of Parametric and Nonparametric Statistical Procedures, 4th edn. Chapman and Hall, New York
Siau K (1999) Information Modeling and Method Engineering: A Psychological Perspective. J Database Manag 10:44–50
Singer J, Vinson NG (2002) Ethical Issues in Empirical Studies of Software Engineering. IEEE Trans Softw Eng 28:1171–1180. doi:10.1109/TSE.2002.1158289
Sjøberg DIK, Hannay JE, Hansen O et al (2005) A survey of controlled experiments in software engineering. IEEE Trans Softw Eng 31:733–753
Spivey JM (1989) The Z Notation: A Reference Manual. Prentice-Hall, New York
SPSS (2003) SPSS 12.0, syntax reference guide. SPSS Inc, Chicago
Tryggeseth E (1997) Report from an Experiment: Impact of Documentation on Maintenance. J Empir Softw Eng 2:201–207
Vegas S, Juristo N, Moreno A, et al. (2006) Analysis of the influence of communication between researchers on experiment replication. Proceedings of the ACM/IEEE international symposium on Empirical software engineering (ISESE'2006). pp 28–37
Verelst J (2004) The influence of the level of abstraction on the evolvability of conceptual models of information systems. Proceedings of the International Symposium on Empirical Software Engineering (ISESE'04). pp 17–26
Wieringa RJ (2003) Chapter 8 - Entity-Relationship Diagrams. In: Wieringa RJ (ed) Design Methods for Reactive Systems. Morgan Kaufmann, San Francisco, pp 77–88
Wohlin C, Runeson P, Host M et al (1999) Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Boston

Ana M. Fernández-Sáez has a MSc in Computer Science from the University of Castilla-La Mancha, Ciudad Real, Spain (2009). She is member of the Alarcos research group and Ph.D student at the Department of Technologies and Information Systems at the same university. Part of her PhD research is done at Leiden University, Leiden, The Netherlands. Her research interests include: UML model quality, quality in model-driven development, software measures and empirical software engineering. Her contact email is: anamaria.fernandez.saez@gmail.com

Marcela Genero is Associate Professor at the Department of Technologies and Information Systems at the University of Castilla-La Mancha, Ciudad Real, Spain. Accredited by ANECA as Full Professor since January of 2012. She received her MSc degree in Computer Science in the Department of Computer Science of the University of South, Argentine in 1989, and her PhD at the University of Castilla-La Mancha, Ciudad Real, Spain in 2002. Her research focuses on the following areas: empirical software engineering, software quality, quality models, conceptual models quality, software modelling effectiveness, gamification in software engineering, etc.). Marcela Genero has published more than 100 peer review papers in prestigious journals (DKE, ESEM, ACM TOSEM, IST, JSS, SOSYM, etc.) and conferences (CAISE, E/R, MODELS, ISESE, METRICS, ESEM, EASE, etc.). She co-edited the books titled "Data and Information Quality" (Kluwer, 2001) and "Metrics for Software Conceptual Models" (Imperial College, 2005), among others. She participated in several program committees (CAISE 2005, METRICS, 2004-2006, ICSM 2007, ESEM 2007-2014, EASE 2008-2014, etc.) and as reviewer of several journals as well (EMSE, IEEE TSE, SOSYM, JSS, IST, etc.). She has organised several conferences, workshops and tutorials on empirical studies in software modelling, evidence-based software engineering, quality in conceptual modelling, etc. She has managed several research projects which involved universities and private companies as partners, related to topics within the research areas previously mentioned. She is member of the International Software Engineering Research Network (ISERN) since 2004.

Danilo Caivano graduated at the University of Bari Aldo Moro, where he also obtained his PhD in 2002 and is currently assistant professor. He carries out his research in the Software Engineering Laboratory at the Department of Informatics. His research and teaching activities focus on topics related to Software Engineering with emphasis on Project and Process Management in collocated and distributed contexts and on software development, maintenance and testing. He is member of the editorial board of several prestigious international journals and conferences. Since 2007 he is Chief Executive Officer of SER&Practices (www.serandp.com), a Spin Off company of the University of Bari that he has contributed to start up. He has managed several large and complex projects, many of which are focused on research and development in partnership with Universities, Research Centers, as well as national and international companies. He is actively involved in the Project Management Institute - Southern Italy Chapter (www.pmi.org) and in the International Software Engineering Research Network (isern.iese.de).

Michel R.V. Chaudron is full professor and head of the Software Engineering Division at the joint department of Computer Science and Engineering of Chalmers and Gothenborg University in Sweden. Prior to that he worked at Leiden University and TU Eindhoven in The Netherlands. From 1997 to 1999, he worked with IT-consultancy CMG (now CGI) in the field of Transport Telematics. He obtained his M.Sc. and Ph.D. degrees from Leiden University, The Netherlands. Prof. Chaudron's research interests are in software architecture, component-based software engineering, software design and software modeling and model-driven software development with a special interest in empirical studies into the effectiveness of modeling. He has published more than 100 peer reviewed papers. He has been active member of several conference in these areas including: CBSE, MODELS, Euromicro-SEAA, ASE, and ESEM.