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

Fernandez Saez, A.M.

CHAPTER 7. CONCLUSIONS

This chapter presents the conclusions of this PhD Thesis, and is organized as follows: Section 7.1 provides a summary of the main findings obtained after carrying out the research developed during this PhD Thesis. The initial research questions and the achievement of the proposed research goals are also analyzed. Section 7.2 outlines the main contributions of this PhD Thesis and its implication for academia and industry, while Section 7.3 states some research lines that are pending for further research. Finally, Section 7.4 lists the whole set of papers published throughout the realization of the PhD Thesis.

7.1. ACHIEVEMENT OF GOALS AND SUMMARY OF FINDINGS

This section reviews the goals established in the Introduction of this PhD Thesis and analyses the way in which they have been achieved.

The main objective of this PhD Thesis was to answer the research questions explained in the Introduction chapter. The following sub-sections show how each Research Question was answered.

7.1.1. RQ1. What is the perception of professionals in industry as regards the value of using UML in software maintenance?

This question helped to provide maintainers' real opinions as regards how UML diagrams (and documentation in general) enable them to perform their maintenance tasks, among other issues. It also helped to establish what kind of companies are fruitfully using UML diagrams as part of their software documentation.

We performed a survey with 178 software maintainers from 12 countries, presented in Chapter 5. The results of the survey revealed some remarkable software maintenance practices, which are summarized in the following paragraphs.

The use of a graphical notation as a complement when attempting to understand the system that will be maintained and to support the design of the changes related to software maintenance tasks is widespread. When a graphical notation is used, the notation selected in the majority of the cases is UML.

As expected, the UML diagrams that are used most frequently during software maintenance are class diagrams, use case diagrams, sequence diagrams and activity diagrams.

Maintainers do not always use the available documentation (with or without diagrams), and use only the source code and its comments. This is owing to problems concerning a lack of synchronization caused by non-updated documentation/diagrams.

If we focus on issues regarding the process, it will be observed that the origin of these UML diagrams is almost always the development phase; i.e., they are not created especially during the maintenance, except when UML diagrams from the software development are not available (in which case they are created specifically during maintenance). Performing reverse engineering techniques is difficult, despite the fact that there are automatic reverse engineering tools. But after the automatic

process, a manual process is required during which the architect attempts to "clean" the diagrams.

- It was detected that some experience is needed in the ICT field in general before working in the software maintenance field. This may be owing to the need to have sufficient experience as regards understanding how systems are built before being able to modify them (see Figure 7.1).

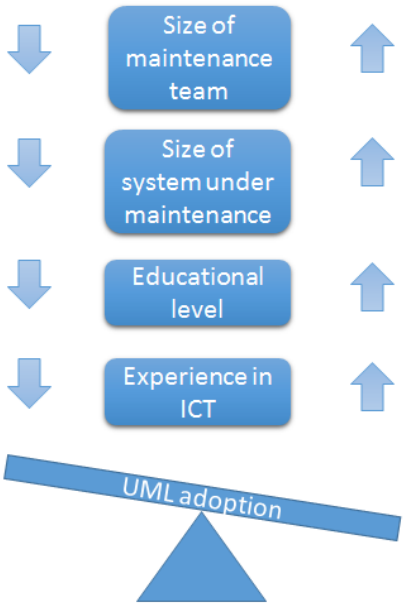

Figure 7.1. Contextual factors that affect the adoption of UML diagrams.

- It is also important to note that most UML users are highly-educated and experienced. It would appear that a higher educational level leads to an increased (likelihood of) use of UML diagrams.
- The size of the system being maintained seems to be an influential factor, because the results obtained show that UML diagrams are extremely popular when the team has to maintain a very large system. This would appear to be logical, since one of the reasons put forward for using UML (or models) is that it helps manage large and/or complicated systems.
- The size of the maintenance team also appears to influence the use of UML. Larger teams use UML diagrams more frequently, proportionately, perhaps because of the improvement to the understanding of the system provided by UML diagrams and owing to the need to share/communicate knowledge in this kind of teams. This is also because small teams use fewer UML diagrams than large ones, since they have facilities for face-to-face meetings and, therefore, require less supporting documentation. It is important to note that larger teams usually work on large (and thus complex) systems, signifying that there is more need for abstraction/managing complexity, and UML assists in this respect.

The survey also highlighted several perceived benefits of using UML during software maintenance, which are related to modeling purposes:

- The main reason for using UML is that less time is needed for a better understanding of the system being maintained; this improves the detection of defects.
- Moreover, when UML diagrams are available as part of the documentation, less effort is required to consult them.

If we focus on the types of tools used to manage (read or modify) UML diagrams, it is quite surprising that the UML Case tools and non-specific UML design tools are used to the same extent. One reason for the frequent use of a non-specific tool might be that a lot of companies work with it for other purposes; employees already have the tool installed, signifying that it is easier to use it and to share diagrams (although they do not check the correctness of the syntax of the diagrams). It is also important to note that some UML users do not use a modeling tool because they manage diagrams on physical paper or blackboards, in which case the diagrams are not digitalized or they are stored as non-modifiable images. The variety of practices related to what tool is used could be a reflection of the fact that different tools are used for different needs: for example, a UML CASE tool could be used to check syntax or to attain a higher traceability diagram-code, while a non-specific tool could be used for a quick brainstorming.

7.1.2. RQ2. Does the level of detail (LoD) of UML diagrams influence software maintenance?

As stated in the introduction, it would also be desirable to help organizations to define the proper LoD required to update the UML documentation in order for it to be synchronized with the source code and thus attain understandability benefits during further maintenance. In order to help organizations to establish a baseline LoD for their UML diagrams, we carried out a family of experiments with 81 students in order to discover the appropriate LoD for UML diagrams. This family of experiments is presented in Chapter 3.

We found that the preferred LoD depends on the purpose of the diagram: there is a slight tendency in favor of using high LoD diagrams to understand the system, and a low LoD to maintain it.

Although high LoD diagrams are preferred when understanding a system that is being maintained, the participants stated that they had more difficulties when reading high LoD diagrams as compared to low LoD diagrams. This can be explained by the fact that the structural complexity of a software diagram affects its cognitive complexity and maintainers, therefore, like a higher LoD because it provides them with more detailed information about the actual implementation details. But the higher amount of (implementation) details also makes the model more difficult to understand.

Despite the fact that the participants stated that they experience more difficulties when reading high LoD diagrams, the majority of them considered that the presence of

some elements represented in the high LoD versions of each diagram is very valuable: attributes and operations in class diagrams and formal messages with parameters in sequence diagrams.

When the domain is well known, or the maintainers are already familiarized with it, this minimizes the time taken to read the UML diagrams and they may work directly on the source code. In conclusion, when the system maintained is small and the application domain is simple and is a well-known domain, then novice software maintainers do not effectively require the use of UML diagrams or other types of technical documentation, independently of their LoD, in the context of maintenance projects for Java systems. We therefore assume that, in this context, the use of UML diagrams does not increase the performance of maintenance operations and might even distract the participants while performing comprehension and modification tasks.

In other contexts, such as when the systems maintained are large and/or technically complex, novice maintainers prefer to use the high level design diagrams/documentation. They use the diagrams in order to roughly understand the system architecture and thus which components/subsystem they have to understand and modify. Then, when the system is understood, they prefer to use low LoD diagrams when performing the maintenance tasks.

Although the results of this family of experiments are encompassed in the case of novice maintainers when they are modifying small and well-known domains, we assume that understanding is also improved in medium to large systems and/or those from unfamiliar domains because they are the environments that benefit most from the presence of a proper (UML) design. However, the investment in the maintenance of the UML documentation required for an appropriate maintenance of a system should not be high.

7.1.3. RQ3. What is the impact of using UML in software maintenance in terms of its use in a software project?

There is not much empirical evidence on the impact of UML modeling in software maintenance in an industrial context. It is sometimes difficult to measure the effect of the UML notation in a real software project, in which there are many uncontrolled variables that might influence the results. This led us to make our first attempt to obtain an industrial point of view concerning the influence of UML on maintenance tasks (for example, the perceived benefits and hurdles of UML modeling practices) through the use of a case study (see Chapter 6) in order to. The data used in the case study was collected at the ICT Department of a multinational transport company in Western Europe. This company is about 100 years old, but we were able to trace the use of ICT in the company back to (at least) the late 1960's. The ICT department has between 800-1000 employees, and plays an essential role in this company's competitive advantage within its sector. This provides an industrial point of view concerning the influence of UML on maintenance tasks.

Firstly, during this case study we detected that UML modeling is commonly used at a low level of formality. The main reason for this is that the purpose of UML

models is communication and in particular the communication of an "overview of the system", which drives users to focus on key parts of the design. Moreover, models in project documentation should be understandable for all the audiences of the documentation.

After detecting why UML is used, we focused on the perceived costs of modeling. The main cost-factors mentioned as regards the use of UML relate to the changing and migration of existing work practices towards using UML, rather than to the actual use of UML. These factors include: the cost of training, the cost of migrating documentation and the cost of changing processes. What is more, the cost of tools is perceived to be a high cost factor for large companies, but this perception is not necessarily so. With regard to the costs, we can also consider some of the hurdles detected as regards the adoption of UML modeling, from the general to the specific. It is first necessary to highlight some hurdles as regards maintaining documentation by itself (non UML).

The hurdles when maintaining UML as part the documentation are the following:

- The most frequently mentioned hurdle was time-pressure, but unfortunately projects always have time pressure so this should not be used as an excuse and projects need to reserve time for a proper management of the modeling.
- Another hurdle is that there is often no clear definition of who is responsible for updating documentation. The updating is further complicated by the fact that duplicates of documentation have sometimes been stored in different archiving systems.
- The difficulty of introducing/migrating to a UML modeling practice was also mentioned, but this is not a specific hurdle of UML and occurs with any migration approach.

These factors make the use of (UML) documentation unnecessarily complicated to manage.

We shall now focus on the hurdles that are directly related to UML modeling alone:

- It is difficult to find a proper level of abstraction and level of detail in order to use UML models in the documentation.
- The difficulty of keeping UML models and the implementation synchronized was also frequently mentioned.

Although the current generation of UML modeling tools does not yet support all these features, it is technically possible to develop a tool with which to solve the majority of the aforementioned hurdles.

If we now concentrate on the beneficial aspects, one perceived benefit is a reduction in the execution time of a project. Some process benefits were also highlighted, such as: improved communication, the prevention of knowledge evaporation and the easing of the diagnosis of problems (especially when using behavior models). UML modeling is also considered helpful as regards improving the

design before starting implementation activities (through an increased ease of peer-reviewing).

If we focus on the product quality benefits of using UML modeling, the majority of maintainers believe that it improves the quality of the ultimate software product. This increased quality might be the result of some of the other benefits mentioned, such as an increased understanding of the system to be maintained and the possibility of monitoring whether an implementation conforms to a design. Finally, in general, structural models like class- and component-diagrams are believed to have a positive impact on the (quality of the) structure of the system (modularity, layering). Modeling, therefore, contributes to the achievement of a good system structure, and a good system structure is known to be a major benefit as regards the maintainability of systems. What is more, behavioral models (sequence diagrams) share most of the generic benefits of modeling, but also aid in the diagnosis of errors.

Finally, we should stress that the benefits and costs/hurdles are difficult to quantify because they are intangible: no company keeps a record of miscommunication, poor early design choices, out-of-date or unavailable documentation and their associated costs. One of the reasons why such benefits are hard to quantify is that modeling is often one of many ways (employed in conjunction with others) of achieving a particular goal. Faults cannot, therefore, then be traced back to a single cause.

7.1.4. RQ4. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?

Projects sometimes use Reverse Engineering (RE) when there is no Forward Designed (FD) documentation available, or as a way around having to spend time on creating FD diagrams. In order to discover what is more beneficial for software maintainers (RE or FD diagrams), a family of experiments with in total 169 students was performed (see Chapter 4). The results obtained revealed that:

- Software maintainers were a little more efficient and effective when using FD UML diagrams that were made by hand during the design phase. This indicates that FD diagrams may, to some extent, improve the maintenance of source code.
- Only in the case of junior maintainers was a better efficiency obtained when using the RE diagrams. This might be explained by the fact that RE diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams.
- In general, those who received FD diagrams experienced fewer difficulties when reading the diagrams when compared to those who received the RE diagrams. RE sequence diagrams were simultaneously found to be less useful than FD sequence diagrams. These participants pointed out that these RE sequence diagrams are not very useful owing to their low level of readability/high level of details. A possible explanation for this is that human engineers make (implicit) assessments regarding which information is important in order to understand key aspects of the system and thus regarding which information should be included in a diagram. We believe that the findings of our experiments indicate that FD class diagrams provide a more

attractive balance between detail and relevant information than RE class diagrams. But, as stated previously, UML diagrams are not usually updated during maintenance tasks owing to the time constraints in industrial environments. The high level use of class diagrams observed during this family of experiments therefore leads us to recommend that companies keep UML diagrams up to date in order to improve maintainers' performance. Sequence diagrams are used less often than class diagrams, probably because of the nature of the tasks (perfective maintenance tasks), and because it would appear that current automated reverse engineering methods are not able to filter out the relevant information in a sequence diagram when employed for maintenance tasks. These factors lead maintainers who are using RE diagrams to work directly with source code.

- Class diagrams are important artifacts that are widely used and highly appreciated by maintainers. However, there is a lack of clarity as to whether the documentation, and UML as part of it, is up-to-date. This goes against an effective use of the diagrams.

7.2. CONTRIBUTIONS OF THE PHD THESIS

The main contribution of this PhD Thesis is the enrichment of the SE body of knowledge concerning the impacts of UML modeling on software maintenance. The different empirical studies performed provide different points of view (academic lab, industry, and company) regarding the factors that influence the use of UML and its impact on software maintenance. The main factors studied in each empirical study (survey, experiments and case study), and the main relations among them are summarized in Figure 7.2. In particular, the realization of this empirical research has allowed us to clarify common beliefs or theoretical assumptions related to the impacts of UML modeling by assessing those assumptions in academic contexts and in different projects in a real industrial context.

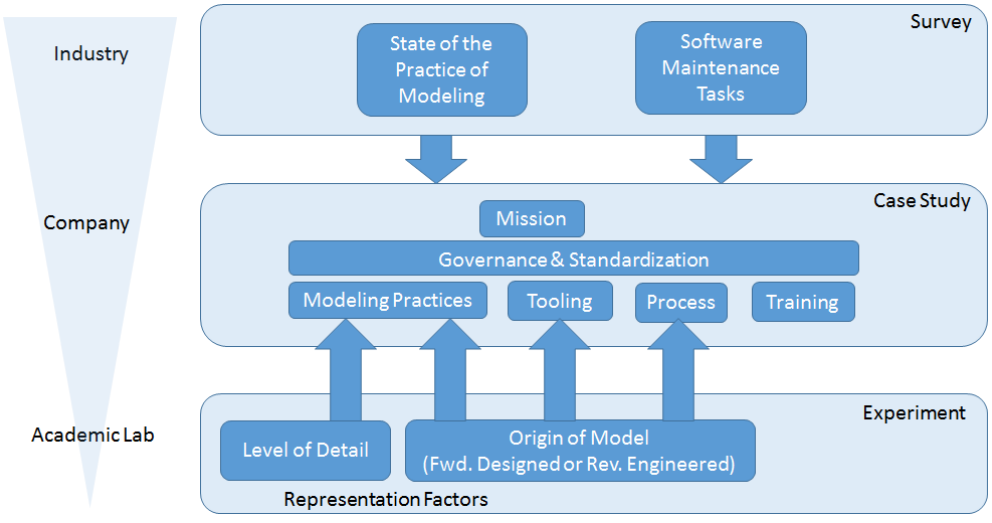

Figure 7.2. Main factors studied and relations among them.

In essence, the contribution of this PhD Thesis is two-fold. First, this study provides sound empirical evidence about the potential benefits of UML modeling in software maintenance. The findings of this study may also contribute to the body of knowledge, particularly in the field of software engineering. Additionally, and from a research perspective, we consider that this study is a milestone towards a more comprehensive understanding of the role of modeling in software maintenance.

The knowledge obtained throughout the realization of this PhD Thesis and its practical implications are summarized below:

- We detected that following a modeling approach, and especially UML, might be part of the approach to documentation, and also part of the approach to implementation. But going further, the UML modeling approach could be considered as a "bridge" between documentation and implementation. In addition, the elements of this UML modeling approach which were influenced by the decisions made as part of the software engineering approach have been highlighted (Figure 7.3). Although scholars have, in some respects, studied the use of modeling notations within software maintenance teams, our knowledge remains fragmented by the divergent efforts that are based on and contribute to theoretical perspectives. This PhD Thesis contributes with a more concrete and less fragmented theory regarding the use of UML.

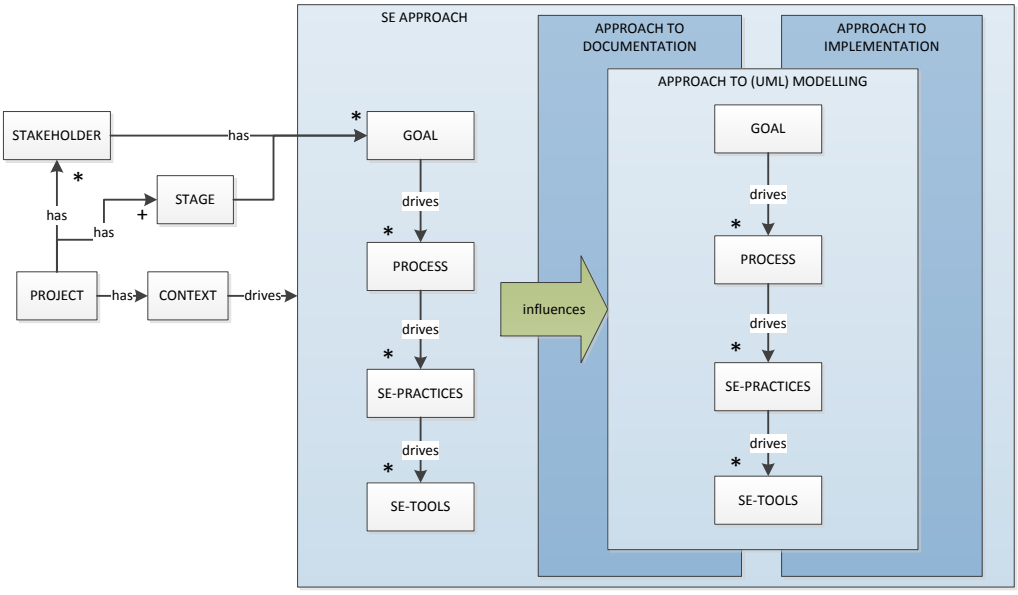

Figure 7.3. Overview of the impacts studied on this PhD Thesis.

- It is very important to target the appropriate level of detail in UML models according to their purposes. In particular, UML models that serve as guides for implementation usually require a high LoD. This high level would be beneficial for those maintainers who are not involved in the development of a specific system and who need to understand it in order to maintain it. When the system is already

understood, a lower LoD is recommended, when UML diagrams are used to perform maintenance tasks or when communicating design decisions. But it is essential that the UML documentation be synchronized with the source code, even though this may be achieved by means of an abstraction step.

- The use of low LoD diagrams is better in comparison to that of high LoD diagrams when performing maintenance tasks (once the system being maintained is understood). This statement is also supported by Briand et al.'s framework (Briand et al., 2001b), 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. It is useless to invest too much in software modeling so as to provide maintainers with very detailed information.
- In order to achieve quality and productivity improvements through the use of UML in software maintenance, we recommend enforcing the practice of creating UML diagrams during the software development, and maintaining them (and the rest of the documentation) in correspondence with the source code. When UML diagrams from the development phase are not available, we recommend generating them using a Reverse Engineering technique. But we also would recommend revising those RE UML diagrams in order to clean and clarify them for a proper understanding.
- Since two key purposes of UML models are communication and attaining an overview of the system, both the parties that produce documentation and those that consume documentation should agree on a level of abstraction and level of detail of UML/documentation that best suits all parties. All parties should be involved in the creation of the UML models and have interactive meetings in order to avoid any possible misunderstandings about their content.
- In software maintenance projects it is necessary to clearly define the responsibility regarding updating/maintaining the (UML) documentation. It should be clear who needs to do this, when this needs to be done, and how. Such practices can be embedded in modern agile approaches – by, for example, including updated documentation in the 'definition of done'. Establishing a versioning method would also be desirable.

The second contribution of this PhD Thesis is related to modeling practices. This PhD thesis provides recommendations concerning best modeling practices when using UML. These recommendations are based on both the empirical data obtained from an industrial case study and expert opinions. This study, therefore, essentially serves as a guide and as a justification for software engineers to continue using (and updating) the UML models of software systems, and to obtain the maximum benefit from them. A list of our main recommendations is summarized below:

- We recommend that efforts should be made to design the UML models during the software development phase in order to have all the benefits of the modeling available during maintenance (apart from the benefits that could be obtained in the development). Following a model-centric approach as one of the companies' software practices improves the understanding of the system and source code.

- The use of modeling and documentation becomes more useful in the case of teams of 6-8 people or more, and also for projects that last longer than 1year.
- In order to obtain the aforementioned paybacks in the maintenance phase, during the development of a system it is important to:
  - Reflect on and define the purpose of using UML, and also define which diagrams are going to be used for each purpose.
  - Involve developers in the software design process.
  - Do not throw documentation 'over the wall', but have interactive (face-to-face) meetings in which the models are explained and questions can be asked about it.
  - The architect should send the model to the team, particularly in offshored projects. That team should then review this and provide the architect with feedback to be checked (all using UML).
  - It is also important not to construct overly long documents or unstructured documentation. This will help maintainers to search for pieces of relevant information.
- In an ideal software project, the development of a software system begins by creating UML diagrams, in addition to keeping them up-to-date during the maintenance, thereby making it easier to perform maintenance tasks. As mentioned previously, it is recommendable to maintain the documentation, and especially the diagrams, in order to obtain all their possible paybacks, even when the system being maintained is well known. During the maintenance it is important to:
  - Ensure that the documentation gets updated. We recommend the use of a diagram-version management tool and taking care when planning for the creation and updating of the documentation, reserving time in the project to update documentation. It is also very important to provide clear criteria that define who should update the documentation, along with when and how it should be updated.
  - Decide on a maintenance/migration policy for models (and documentation). When migrating legacy software, pay particular attention to when diagrams/models are introduced. Special guidelines may be useful to standardize which diagrams to introduce and, if making separate additions to existing documentation, where new types of documents are stored in repositories/file systems.
- The following list of recommendations applies to both development and maintenance:
  - Integrate peer-reviews for models and documentation in the development process.
  - Invest in training people how to use the modeling tools, showing not only their main features but also their details.
  - Train those software engineers who are not familiar with the advantages and who do not know any possible positive aspects of a change towards using modeling in their process, thus enabling them to become aware of the long-term benefits of using modeling languages (especially UML).

- Conventions for naming models, classes, methods and attributes should be defined.
- Complement the diagrams with personalized legends in order to clarify the non-UML or non-standard parts of the diagram.
- Define the level of detail which should be presented in diagrams. This would help to avoid an excess of documentation, or a lack of it.
- Ensure that the producers and consumers of documentation know each other. This would thus enable them to agree on standardizing the aforementioned issues, and also to agree on the relevant information to be documented and the level of abstraction which is needed in each case.
- Some standardization is also needed at project and organizational level in order to solve problems related to the accessibility of documentation, always using the same project structures.
- Use tooling for the automated checking of UML, e.g. a consistent use of patterns, design principles, naming conventions, or correspondence of source code with design. These may be applicable for long-lived projects that have a high maturity (in order to increase the completeness and correctness of the documentation).
- Use tools that support searches in models.
- Use tools with functionalities related to the traceability between model and text so as not to leave the documentation and model out of synch.
- Make modeling tools available from the start of the project in order to obtain their benefits from the early stages in a project.
- Finance tooling centrally – not at the project level. This prevents projects from trying to circumvent modeling by avoiding (generally small) tooling costs.

7.3. FUTURE RESEARCH LINES

The findings obtained are a first approach to discovering the contextual factors that affect the effective use of UML during software maintenance in industrial projects. We believe that we have now taken the first step, but this topic deserves to be studied in greater depth. Possible future directions for this line of research are:

- It would be interesting to analyze the effect of different UML notations by, for example, comparing formal diagrams with box & lines. Several reasons (such as time pressure or the lack of training on notation) cause maintainers/developers to use informal box and line (UML) diagrams. It would be interesting to investigate the impact of this relaxed use of UML on the understanding and maintenance of a software system.
- Developing supporting tools for a proper UML documentation. The current tools are focused only on managing the UML diagrams, and they include features which help to create correct UML diagrams (such as checking correctness, helping to manage consistency, etc.). But it would be desirable to have a tool with which to create and maintain documentation containing a mix of both text and diagrams. It would be important to have a tool with functionalities that improve the traceability between model and text so as not to leave the documentation and model out of

synch. It also would be useful to have a tool that supports a correct versioning system of the diagrams, supports searches in models, or a tool capable of presenting different views (e.g. at abstraction levels or in detail) of the same diagram adapted for different purposes (for different consumers of this information).

- An empirical investigation regarding whether this extra effort when creating and maintaining UML documentation is worthwhile is also still pending. This would provide the software community with the knowledge regarding the return on investment of UML modeling in software maintenance. This topic has not been explored in detail, and this is necessary if we wish to increase the acceptation and usage of UML in industry.
- It is important to strengthen empirical validation in order to mitigate the aforementioned threats to the validity of the empirical studies presented in this Thesis. It would, therefore, be interesting to follow the same research line of this thesis and carry out replications of these studies with different configurations or different cases. The families of experiments could be replicated by considering realistic software systems related to larger and unknown domains in order to verify whether the findings obtained are still valid. It would also be interesting to replicate these laboratory studies with industrial practitioners. With regard to the case study, it would be interesting to replicate the research by conducting more industrial studies with different software systems (e.g., different programming languages, technologies, size, complexity, etc.) from different companies, using different software development methodologies or at least those from different domains.

7.4. PUBLICATIONS

As mentioned in the Introduction chapter, this PhD Thesis is presented as a collection of ten papers, published in diverse journals, conferences and books related to Software Engineering.

Other works, apart from the 10 selected to form the chapters of this thesis, have also been published. A complete list of all the papers published throughout the period in which this PhD thesis was being completed is provided below.

Publications in national conferences

1. Fernández-Sáez, A. M., Genero, M., & Romero, F. P. (2010). SLR-Tool: A Tool for performing systematic literature reviews. *XV Jornadas de Ingeniería del Software y Bases de Datos* (JISBD'10), Valencia, Spain.
2. Fernández-Sáez, A. M., Genero, M., & Chaudron, M. R. V. (2010). Investigating the benefits of UML on software maintenance: A research proposal. *9th edition of the BElgian-NEtherlands software eVOLution seminar* (BENEVOL 2010), Lille, France.
3. Fernández-Sáez, A. M., Genero, M., & Chaudron, M. R. V. (2011). A research plan for gathering empirical evidence of the benefits of using UML in software maintenance. *XVI Jornadas de Ingeniería del Software y Bases de Datos* (JISBD'11), La Coruña, Spain.

4. Genero, M., Piattini, M., & Fernández-Sáez, A. M. (2011). Revisión sistemática sobre el aseguramiento de la calidad de requisitos. *12th Argentine Symposium on Software Engineering* (ASSE'11), Córdoba, Argentina.

Publications in international conferences/workshops

1. Fernández-Sáez, A. M., Genero, M., & Romero, F. P. (2010). SLR-Tool: A tool for performing systematic literature reviews. *5th International Conference on Software and Data Technologies* (ICSOFT'10), Atenas, Greek.
2. Fernández-Sáez, A. M., Genero, M., & Chaudron, M. R. V. (2011). Empirical investigation on the benefits of using UML in software maintenance. *12th International Conference on Product Focused Software Development and Process Improvement* (PROFES'11) – Doctoral Symposium, Torre Canne (BR), Italy.
3. Fernández-Sáez, A.M., Genero, M., & Chaudron, M.R.V. (2012). Does the Level of Detail of UML Models Affect the Maintainability of Source Code? *Experiences and Empirical Studies in Software Modelling* (EESSMod'11) Workshop at MODELS 2011, Wellington, Nueva Zelanda. LNCS 7167, pp. 133–147.
4. Fernández-Sáez, A.M., Hendriks, P., Heijstek, W.,& Chaudron, M.R.V. (2012). The Role of Domain-Knowledge in Understanding Activity Diagrams – An Experiment. *Experiences and Empirical Studies in Software Modelling* (EESSMod'12) Workshop at MODELS 2012, Innsbruck, Austria. MODELS 2012 Workshop, pp. 133–147.
5. Fernández-Sáez A.M., Genero, M., Chaudron, M.R.V., & Ramos, I. (2013. Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?: A Controlled Experiment. *International Conference on Evaluation and Assessment in Software Engineering* (EASE 2013), Porto de Galinhas, Brazil, pp. 60-71. (One of the best papers of the conference award).
6. Fernández-Sáez A.M., Chaudron, M.R.V, & Genero, M. (2013). Exploring Costs and Benefits of Using UML on Maintenance: Preliminary Findings of a Case Study in a Large IT Department. *Experiences and Empirical Studies in Software Modelling* (EESSMod'13) Workshop at MODELS 2013, Miami, Florida, USA.
7. Fernández-Sáez, A.M., Caivano, D.,Genero, M., & Chaudron, M.R.V. (2015). On the use of UML documentation in software maintenance: Results from a survey in industry. *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (MoDELS 2015): 292-301.

Journal publications (included in the JCR)

1. Genero, M., Fernández-Sáez, A. M., Nelson, H. J., Poels, G., &Piattini, M. (2011). A Systematic Literature Review on the Quality of UML Models. *Journal of Database Management* 22(3): 46-70. (**JCR Index 2011 = 0.875**).
2. Fernández-Sáez, A. M., Genero, M., & Chaudron, M. R. V. (2103). Empirical studies on the influence of UML in software maintenance tasks: A systematic literature review. *Information and Software Technology*. (**JCR Index 2013 = 1.25**).

3. Fernández-Sáez A.M., Genero, M., Chaudron, M.R.V., Caivano, D., & Ramos, I. (2015). Are Forward Designed or Reverse-Engineered UML diagrams more helpful for code maintenance?: A family of experiments. *Information & Software Technology* 57: 644-663(**JCR Index 2015 = 1.046**).
4. Fernández-Sáez, A.M., Genero, M., Caivano, D., & Chaudron, M.R.V. (2016). Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments. *Empirical Software Engineering* 21(1): 212-259 (**JCR Index 2016= 3.275**).
5. Fernández-Sáez, A. M., Chaudron, M. R. V., and Genero, M. An industrial case study on UML modelling practices and their effectiveness in software maintenance. Empirical Software Engineering (**JCR Index 2016 = 3.275**).