Chapter 1

Introduction

In this chapter, we present the contexts, motivations, and goal of this study. We also outline our approach, and summarize the main findings and contributions of this study. After reading this chapter, readers should have a sufficient background about the problem domains and how we attempt to address them.

1.1 Quality in Modeling

Modeling is an activity to create representations of the domain of software systems, which involves creating technical solutions that meet the systems' requirements. Good software models perfectly capture user requirements and translate them into technical designs. However, a perfect match to requirements is not the only aspect that is crucial for software models. Models of systems are also characterized by their syntactic and semantic quality.

Syntactic quality of models indicates the conformance of models to the standard specification of the modeling language. This means that the use of symbols or notations to represent ideas, concepts, or behaviors must comply with their specified usage. On the other hand, semantic quality concerns the meaning of representations of models. The semantic quality of models can be determined from the degree to which they correctly and unambiguously represents concepts or behaviors being modeled. In addition to these quality aspects, Lindland et al. proposes another quality aspect, namely pragmatic quality [82]. The main goal of pragmatic quality is comprehension, which can be achieved by selecting the most appropriate way of expressing a single meaning.

Paying attention to the quality of software models is important because it might determine the quality of the final software product. This is particularly true, however, when software models are used as a foundation to develop the actual software system. If a software model does not meet the requirements, there is a big chance that a wrong product will be delivered. The same applies if software models have a poor syntactic or semantic quality—although in this case wrong *interpretation* of models would be the main cause of

delivering a wrong product.

In practice, the quality of software models is primarily important for software architects, developers, and maintainers. Software architects, for example, need to assure that high level design decisions are appropriately translated into detailed designs. Software developers need comprehensible and consistent models based on which the implementation code is written. Finally, software maintainers depend on models that have sufficient correspondence with the actual implementation so that they can perform maintenance activities efficiently.

1.2 Problem Statement

There is no doubt about the premise that technical solutions described in software models must meet the system requirements. In fact, a seamless match with requirements is the major concern for software models to have any value. Nevertheless, sound technical solutions must also be described in manners that leave no room for misinterpretations. In this regard, the syntactic and semantic quality of models become crucial factors that might determine whether sound technical solutions will be translated *correctly* into working products.

The main issue to consider if one wants to perform system modeling—not to mention good modeling—is that it requires substantial investments. From an economic point of view, any types of investment must be justified in that they will payback at a later stage. Therefore, in the context of software projects, investing in modeling should be justified by benefits that can be gained later during software development or maintenance, such as improved productivity and improved product quality. When such benefits are not foreseeable, modeling software systems would simply be an expensive luxury without an added value to the system being developed.

The big issue about software modeling actually lies exactly at the assumption that it will deliver quantifiable benefits for software projects. Therefore, the problem is how we can investigate and prove whether or not modeling or quality in modeling brings any benefits during software development. So long as this question remains unanswered, it would be difficult to motivate and justify modeling activities in real software projects.

1.3 Objective of the Study

The goal of this thesis is to investigate the benefits of modeling on the quality of software. In particular, we focus our attention on the Unified Modeling Language (UML) [100] as a modeling language because UML is the most widely used modeling language in industry.

To achieve the objective, five research questions have been defined during this study. The research questions outlined below are formulated to look at the impacts of UML modeling on software quality from different perspectives (e.g., from the point of view of software engineers and from empirical data obtained from industrial software projects). Apart from the different perspectives, new insights obtained during the study have also led to the formulation of new research questions. Several research strategies are employed to address

these research questions. By considering different angles and conducting multiple research strategies in answering the grand question, we expect to obtain a more comprehensive understanding about the impacts of UML modeling on the quality of software.

- RQ1: How is the UML used in practice; and do software engineers perceive any benefits of using UML?
- RQ2: What is the effect of using UML for modeling a software system on quality and productivity?
- RQ3: What is the effect of the quality of UML models on model comprehension?
- RQ4: Does the quality of UML models correlate with the quality of the resulting software?
- RQ5: Can we predict fault-prone software modules or components based on the quality of the software model?

1.4 Research Methodology

Our approach to address the research questions of this study is empirical in nature. Empirical research is a research methodology that relies on direct or indirect observations to explain phenomena. Empirical studies attempt to compare theories with reality and improve the theories as a result [107].

Empirical research has been increasingly used by researchers in software engineering because it allows them to evaluate and improve techniques, methods, and tools in developing software [14, 132]. The results of such evaluation can then be used to determine the adoption of the new tools or techniques, and thus reducing the risk of adopting wrong or ineffective technology [123].

In the context of this study, empirical research methods are useful because they provide means for us to evaluate our assumptions or hypotheses related to the aforementioned research questions. As discussed earlier, it is believed that the quality of software models has positive effects on the quality of the final software products. This assumption or belief concerning the (causal) relationship between model quality and software quality needs to be validated empirically—that is, based on data obtained from industry or based on experimentations. Ideally, assessments of assumptions (hypothesis testing) should be done repeatedly using different observations in order to increase confidence on the obtained findings.

In conducting this study, we use three research strategies, namely survey [109], case study [114], and experiment [130]. In the survey, we study how software developers' use UML. In particular, we are interested to understand the perceived benefits of using UML during software development (RQ1). The case study strategy is used to address RQ2, RQ4 and RQ5 respectively. The case study is an industrial software project from which we collect various data to assess the impact of UML modeling on downstream software development. Finally, we perform an experiment to investigate the effect of the quality of UML models on model comprehension (RQ3).

1.5 Contributions and Outline

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

The second contribution is related to modeling practices. This study provides recommendations concerning best practices of modeling using UML. These recommendations are based on careful analyses of empirical data from an industrial case study as well as expert opinions. Therefore, this study essentially serves as a guide and justification for software engineers to model software systems using UML, and to get the most benefits out of it.

The organization of this work is as follows:

- **Chapter 2: Background.** In this chapter, we briefly introduce UML as a modeling language and its notable characteristics. Additionally, we elaborate on some UML diagram types that are commonly used in practice. Following that, we present some challenges in modeling using UML. Last but not least, we present the state of the art in quality assurance of UML models.
- **Chapter 3: The Use of UML in Practice.** In this chapter, we provide empirical findings from a survey on the use of UML amongst 80 professional software engineers. We explore software engineers' opinions on common styles of using UML and how they perceive the impact of using UML on productivity and quality in software development. One of the results reveals that the impact of using the UML on productivity is perceived mostly in the design, analysis, and implementation phases.
- **Chapter 4: The Impact of UML Modeling on Software Quality.** In this chapter, we explore whether the use of UML helps improve the quality of the resulting software and productivity in fixing defects. While in the previous chapter the reported benefits of using UML are those perceived by software developers, in this chapter we explore the benefits of using UML in an industrial project.
- **Chapter 5: The Impact of Level of Detail in UML Models on Comprehension.** In this chapter, the impact of modeling is further explored by introducing the notion of level of detail (LoD) as a form of style and rigor in modeling. In a controlled experiment, we investigate whether LoD in UML models affects the correctness and efficiency in comprehending UML models. Results show that the effect of LoD in UML models on model comprehension is significant.
- **Chapter 6: Level of Detail in UML Models and its Relation to Defect Density.** In this chapter, we propose some measures to quantify LoD. The proposed measures are applicable to UML class- and sequence- diagrams, and are evaluated using a significant industrial Java system. Based on the case study we have found that LoD of messages in sequence diagrams is significantly correlated with defect density in the implementation.

- **Chapter 7: Assessing UML Design Metrics for Predicting Fault-prone Classes in a Java System.** In this chapter, we evaluate the usefulness of UML design metrics to predict fault-proneness of Java classes. We use historical data of a significant industrial Java system to build and validate a UML-based prediction model. Based on the case study we have found that *level of detail of messages* and *import coupling*—both measured from sequence diagrams, are significant predictors of class fault-proneness.
- **Chapter 8: Conclusions.** In this chapter, we draw conclusions and discuss future work. Additionally, we discuss the contributions of this thesis and recommendations for software engineering practice.