Summary

As the de facto industry standard for software modeling, the Unified Modeling Language (UML) is used widely across various IT domains. UMLs wide acceptance is partly because the language offers flexibility and freedom in modeling software systems: 1) UML provides an extensive set of modeling notations that can be used to model various concepts; 2) UML can be used both in a casual and formal manners.

In the context of model-driven software development, the degree of freedom in which UML is used raises an important issue related to model quality. Different styles and rigors in using UML affect the quality of the resulting models. It is then logical to think that the level of quality of the UML model may affect the quality of the resulting software.

Notwithstanding the increasing role of UML in model-driven software development, there is still little effort devoted to investigate the effects of UML modeling on software development. This issue is particularly interesting if we take into account the way in which UML is used in practice. One of the big questions is: does the way in which UML is used affect the quality of the resulting software? This thesis reports on a series of empirical studies that are aimed at answering such a question. These studies employ different research strategies (i.e., survey, experimentation, and case studies) as to approach the subject matter from different perspectives.

A survey was conducted to investigate the way in which UML is used in practice from software engineers point of view. The results indicate that software engineers generally consider the completeness and level of detail in UML models rather low. Moreover, software engineers consider incompleteness in UML models as a driver to deviate in the implementation. The effect of model completeness was further investigated in an industrial case study. By comparing the quality between components (measured in defect density) that are modeled using UML and those that are not modeled, we have learnt that modeled components have significantly higher quality than unmodeled ones. This finding suggests that completeness in UML models is crucial to achieve a certain level of quality in the implementation. Additionally, the use of UML was found to reduce the effort required to fix defects. These results are observed after controlling for the effects of potential confounding factors, namely class coupling and complexity.

In practice, the level of detail in UML models varies. Level of detail (LoD) represents the amount of information that is used to specify models. The effect of LoD was investigated in an experimental setting involving 53 master students. The results of the experiment

show that higher LoD in UML models increases model comprehension in terms of comprehension correctness and comprehension efficiency. Furthermore, a set of UML metrics was defined to measure LoD in UML models. These metrics were applied to an industrial case study, and the result shows that LoD in the UML model correlates with the quality of the implementation: the higher the LoD, the higher the quality of the implementation. In particular, LoD of messages in sequence diagrams is a significant factor that drives components quality. Another interesting finding is related to the use of UML design metrics as predictors of component fault-proneness. LoD of messages and import coupling (IC) measured from sequence diagrams are significant predictors of class fault-proneness. Furthermore, the prediction model built only using these UML design metrics demonstrates better accuracy compared to the prediction model that was built only using code metrics.

To conclude, this series of empirical studies attempts to address a pivotal question concerning the benefits of UML modeling in software development, particularly from a quality perspective. The results of these empirical studies show that the use of UML provides benefits in terms of increased quality and productivity in software development. The availability of UML models also allows early prediction of defects in software systems. Such prediction is potentially useful for identifying and fixing defects early during software development, and for prioritizing testing.