The effects of UML modeling on the quality of software

Nugroho, A.

Chapter 8

Conclusions

The Unified Modeling language (UML) as the de facto industry standard for software modeling is widely used in software engineering projects. The objective of this study is to evaluate the benefits of using UML in software development in terms of improved software quality. Additionally, this study looks at how UML is used in practice and attempts to unveil subjective benefits from the point of view of software engineers. In this chapter, we revisit and summarize the findings of this study. We also outline future work to follow up emerging questions not covered in this study.

8.1 Summary of Findings

In this section, we summarize empirical evidence that we have observed throughout this study by revisiting the research questions.

8.1.1 RQ1: How is the UML used in practice; and do software engineers perceive any benefits of using UML?

We start our investigation into the benefits of UML modeling in software development by looking at how it is used in practice and by assessing whether software engineers perceive any benefits from using it.

Based on a survey amongst professional software engineers, we have found that UML models in practice are often lacking completeness. This finding is important because model completeness is often considered as one of the major forces that drive software engineers to deviate from UML models in the implementation phase. Additionally, we have learnt that model completeness affects developers' strictness in implementing certain modeling constructs.

Another important finding concerns the preference of using details and applying com-

pleteness in UML models. We have learnt that software engineers prefer giving more attention (i.e., by putting more details or increasing completeness) to parts of models that capture complex or critical system components.

Concerning the benefits of UML in software development, software engineers perceive UML to be most useful to increase productivity in the design-, analysis-, and implementation phase respectively. Interestingly, we also found that engineers who perceive a productivity increase in the implementation phase are those who are in favor of high conformance to UML models. This result suggests that the use of UML increases productivity in the implementation phase if the UML model is followed strictly. Furthermore, we have learnt that UML is perceived to be most effective in improving the understandability and modularity of software systems.

The aforementioned findings have enriched our understanding about the role of UML in software development and the associated challenges. Overall, from a software engineer's perspective we observe that UML as a modeling language has the capacity to help improve both productivity and quality in software development.

8.1.2 RQ2: What is the effect of using UML for modeling a software system on quality and productivity?

The second research question we aim to answer is whether the use of UML can help improve the quality and productivity in developing software systems. To answer this question, we use a case study of an industrial software project.

In the case study, we collect empirical data and carefully evaluate the impact of using UML on the defect density of Java classes and on the effort spent on fixing defects. After controlling for the effects of class coupling and complexity, we have found that the use of UML modeling remains a significant factor that explains the variability of defect density in the implementation classes. More specifically, Java classes that are modeled using UML are found to have a significantly lower defect density than those that are not modeled. This result indicates the potential benefits of UML modeling for improving the quality of software.

Results also show that defects related to behaviors modeled using sequence diagrams require significantly less time to fix than defects related to unmodeled behaviors. This result remains consistent after controlling for the effects of potential confounding factors, which include the number of modified files, the number of persons involved in fixing defects, and defect priority.

In practice, modeling all software modules/classes/components and behaviors might not be possible due to various reasons such as resource and time constraints. However, the findings that we have found in the case study suggest that investing in upfront modeling pays off in terms of increased software quality and productivity in fixing defects. Because the costs of finding and fixing defects increase as a project progresses, upfront modeling should be seen as a measure to prevent defects, which lead to cost-saving—we believe that the cost of modeling is much cheaper compared to the cost of fixing defects caused by the absence of modeling.

8.1.3 RQ3: What is the effect of the quality of UML models on model comprehension?

Having confirmed the effect of UML modeling on the quality of the final implementation, the next research question to ask is whether the *quality* of UML models has any effects on downstream software development. We address this question by performing an experimental study involving 53 graduate students majoring in Computer Science at the Eindhoven University of Technology, the Netherlands.

In the experiment, we introduce one aspect of UML quality, namely level of detail (LoD), and investigate its impact on model comprehension. Having compared two independent experimental groups, we have found that the group receiving a UML model with high LoD comprehends the model better than the group that receives a model with lower LoD. In essence, the results show that applying higher LoD to a UML model significantly improves correctness and efficiency of subjects in comprehending the model.

The evidence that we have found concerning the effect of LoD on model comprehension has shed light on the effect of UML quality attributes on downstream software development. In particular, the evidence shows that increasing LoD in UML models might increase readers' model comprehension, which in turn will improve the correctness of the implemented system.

8.1.4 RQ4: Does the quality of UML models correlate with the quality of the resulting software?

We continue investigating the impact of UML model quality by evaluating the relationship between Level of Detail (LoD) of UML models and defect density in the implementation. To this aim, we use the case study discussed earlier to conduct the investigation.

We propose LoD as a measure of the quality of UML models and use defect density as a measure of the quality of the implementation. We define four LoD measures based on class- and sequence diagrams. Having applied the LoD measures, we found a significant correlation between LoD of messages in sequence diagrams and defect density. The negative nature of the correlation suggests that classes that are modeled in sequence diagrams in which messages are specified using high LoD tend to have a lower defect density in the implementation. LoD in UML models, which represents the amount of information that is used to specify modeling elements, is assumed to affect developers' comprehension in implementing models. Poor LoD in UML models reduces developers' comprehension, which in the end leads to the introduction of defects. The results of the analysis confirm this assumption.

The main lesson learnt from the finding is that level of detail in behavioral diagrams (especially sequence diagrams) is a potential indicator of defect density in the implementation. Hence, software engineers should pay more attention to the level of detail in behavioral diagrams as it might influence the introduction of defects. Furthermore, the proposed metrics can be used as practical measures to assess and maintain the quality of UML models in software projects.

8.1.5 RQ5: Can we predict fault-prone software modules or components based on the quality of the software model?

Having witnessed the usefulness of LoD measures as predictor of defect density in the implementation, we investigate the feasibility of using UML design metrics such as LoD to predict fault-proneness of software modules. Using the same case study, we extract metrics from class diagrams and sequence diagrams of the UML model and from source code to construct three prediction models: 1) based on UML metrics, 2) based on code metrics, and 3) based on UML and code metrics.

Results show that two UML design metrics, namely *message detailedness* and *import coupling*—both are measured from sequence diagrams—are significant predictors of class fault-proneness. Furthermore, we have found that the prediction model built using the combination of UML design metrics and code metrics demonstrates the best performance. Interestingly, we have also observed that using only UML metrics as predictors gives better performance than using only code metrics. Finally, we also learn that the prediction model built using UML metrics turns out to be the most cost effective.

The most important point to note concerning the results is that we again see how certain attributes in a UML model are significantly correlated with the quality of software. This result also shows that we can perform fault-proneness prediction with reasonable accuracy using UML design metrics. In practice, the results of fault-proneness predictions can be used to prioritize verification or testing effort based on the probability of software components to contain faults.

In Figure 8.1 we distill the main findings of this study in terms of a cause-effect diagram. Boxes represent variables studied and arrows indicate the direction and the nature (positive or negative) of the cause-effect relationships between variables. Notice that dashed arrows represent hypothetical effects. We do not have empirical evidence about such effects, but we have valid and strong assumptions about their presence. The chapters in which the confirmed effects are discussed are also indicated in the picture.

Figure 8.1 shows that the immediate benefit of having good quality UML models (e.g., in terms of high degree of level of detail and completeness) is higher system comprehension. We believe that higher system comprehension is a major factor that explains the impacts of UML model quality on developers' conformance to UML models, which later increases productivity in the implementation. We also see in the figure that the positive effects of UML model quality on the quality of the final system and productivity in fixing defects are also due to increased system comprehension. Therefore, in the context of our study we conclude that the benefits of UML model quality in software development can be mainly attributed to the role of UML models in improving software engineers' comprehension about a system and providing guidance on how it should be implemented.

Also notice how some variables other than system understandability affect the relationship between UML model quality and software quality. Developers' conformance to UML models and component complexity and coupling have a strengthening- and weakening effect respectively on the relationship between UML quality and the quality of the implemented system. Apart from that, developers' conformance to UML models increases the degree of

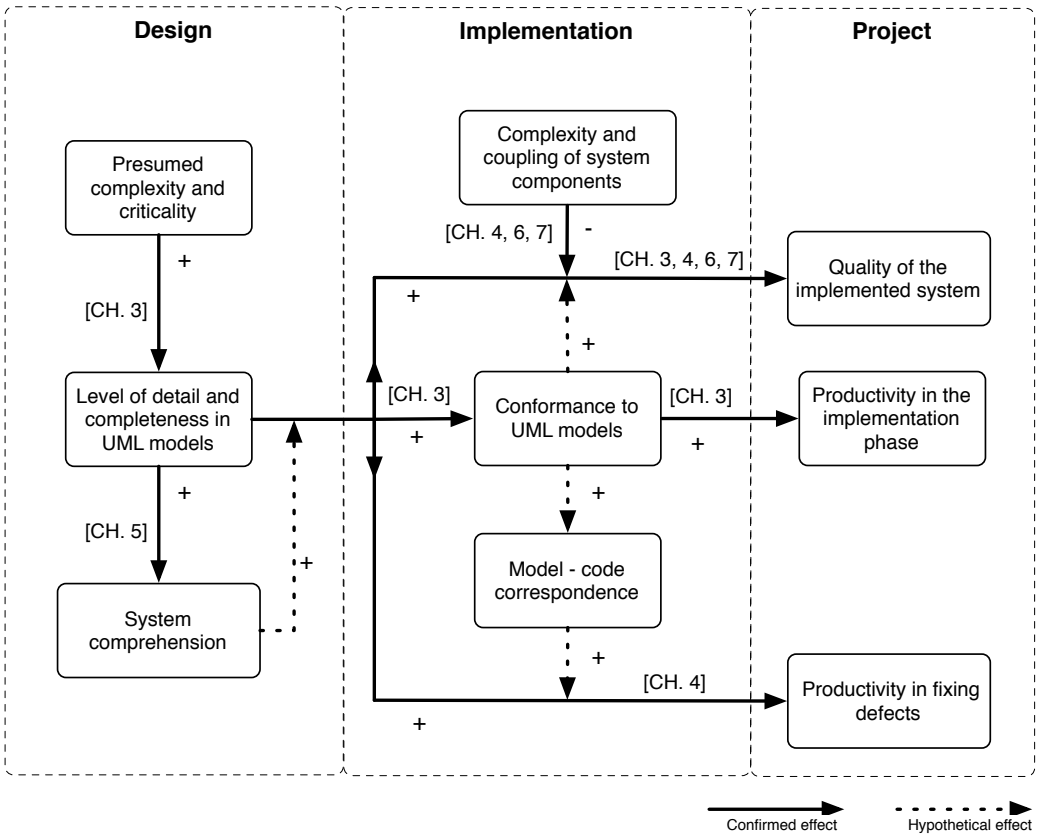


Figure 8.1: Cause-effect diagram summarizing the main findings of the study

model – code correspondence, which later also strengthens the relationship between UML quality and the productivity in fixing software problems.

8.2 Contributions

The contributions of this thesis are summarized as follows:

- **Devising Level of Detail (LoD) metrics as a measure of quality of UML models.** LoD metrics measure the amount of information that is used to describe modeling elements. We have defined LoD metrics for class- and sequence diagram of UML models. Considering the effects of LoD in UML models on model comprehension and on the quality of the final system, measurement of LoD can be incorporated in the design quality assurance activity.
- **Providing tool support for assessing LoD in UML models.** We have provided

a tool that supports measuring LoD in UML models. This support is implemented as an additional feature in the MetricView tool [3]. Tool support speeds up the transfer of the practice of maintaining LoD to modeling practice in industry.

- **Enriching SE body of knowledge concerning the impacts of UML modeling on downstream software development.** This contributes to the body of knowledge concerning the impact of UML modeling on the quality of software systems. In particular, through conducting empirical research we have clarified common beliefs or theoretical assumptions related to the benefits of UML modeling by assessing those assumptions in a real industrial project. Lessons that we have learnt from such assessments improve our knowledge about modeling practice and stimulate follow-up investigations for future research in software engineering.

8.3 Notes on Additional Case Studies

In the course of our study, we have conducted additional industrial case studies besides the one discussed in some earlier chapters—that is, the IPS project. In particular, we have replicated the studies reported in Chapter 4 and 6 using two additional case studies, namely RACE and BHN respectively. These additional case studies are obtained from the same organization. Therefore, in total we have taken into account three industrial case studies to answer some of the research questions. In this section, we briefly discuss the findings from these additional case studies.

Findings from the additional case studies are reported as a master's thesis in [128]. In this report, the three industrial case studies were used to investigate the research questions concerning: 1) the use of UML and its impact on defect density; 2) the relationship between LoD in UML models and defect density of the resulting implementation. The results show that only in one case study (i.e., IPS) we could confirm our hypotheses. In the other two cases studies, we did not observe affirmative evidence. Nevertheless, an investigation was also conducted using a case study of an embedded system from a organization that mainly produces consumers electronics. The results were reported as a master's thesis [50]. The results from the case study, which was called Blue-Ray Middleware, support our findings concerning the effects of using UML that reduce defect density and the effort of fixing defects.

Learning from the discrepancies of the results, we subsequently turned our attention to the characteristics of the case study that shows positive results. We learnt several key facts that distinguish the IPS project from the other projects. Nevertheless, one aspect that really discriminates IPS from RACE and BHN is the degree of model – code correspondence—that is, the extent to which implementation code corresponds to the UML model. While we did not perform formal assessments to measure it, we could rather easily tell the degree of correspondence from how strict class-, method-, and attribute names in the UML model are named accordingly in the implementation. In this respect, no project but IPS demonstrates a sufficiently high degree of model – code correspondence. Additionally, the size of the UML model is an important factor in the statistical analyses. In the RACE project, the number of sequence diagrams is quite small, which resulted in low variability in the metrics being

Table 8.1: Project Summary

Projects	# staff	Duration	Off-shored	Status	Model Size	SLoC
IPS	25 people	2.3 years	India	finished	104 UCs 266 classes 34 CDs 341 SDs	152,017
BHN	18 people	2 years	India	finished	25 UCs 137 classes 28 CDs 115 SDs	135,454
RACE	10 people	1 year	India	finished	9 UCs 44 classes 12 CDs 22 SDs	125,168

measured. This factor hindered the statistical tests to detect meaningful results.

Table 8.1 provides the characteristics of the three case studies. The information about model size and system size gives an insight about the degree of completeness of the UML model in the respective project. The size of the three systems (in KSLoC) is quite similar. The IPS system is the largest in size, while RACE is the smallest. Yet, the difference is only around 25 KSLoC. Considering the average class size in the three systems, which is around 100 SLoC, this difference is roughly equal to 250 classes. While there is no substantial difference in system size, the difference in size of the UML models is more significant. For instance, the IPS project has roughly three and fifteen times as many sequence diagrams as the BHN and RACE project respectively. Hence, in the BHN and RACE projects there were more functionality implemented without technical specifications—that is, the developers were given more freedom to determine how the functionality should be implemented. We suspect this circumstance affects the degree of model – code correspondence in the BHN and RACE projects.

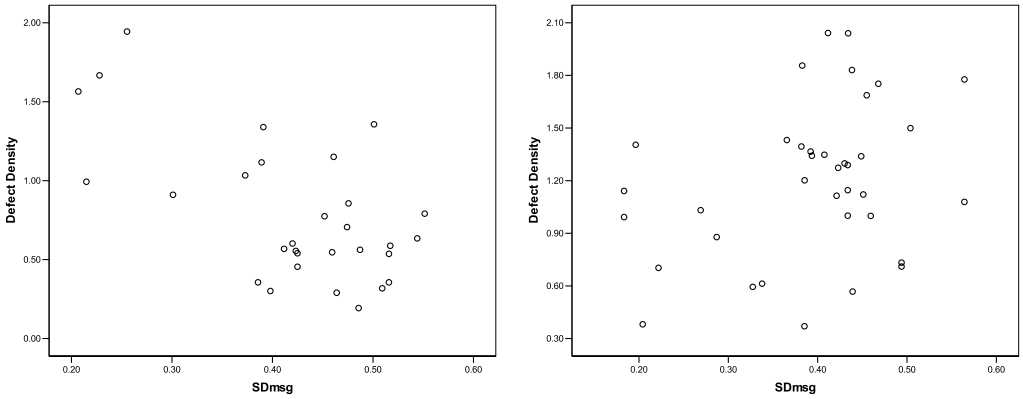
The above argument is supported by the information obtained from an interview with the architect of the BHN project. We learnt that the developers in the project were indeed given quite some freedom to implement the system, including to deviate from the UML model. This is rather surprising given the fact that the amount of details that has been put in the UML model is quite high. Naturally, deviations in terms of static structure generally are more obvious, whereas deviations in terms of dynamic behavior are not immediately observable. Hence, in this case it is possible that the degree of model – code correspondence is actually worse than what we have seen at the surface.

Table 8.2 shows the results of a multivariate analysis that assesses the correlation between SD_{msg} and defect density in the BHN project—the same analysis results for the IPS project was presented in Table 6.8. Table 8.2 shows that after controlling for the effects of coupling and complexity, it was observed that SD_{msg} is not significantly correlated with defect density ($p > 0.05$). Only complexity has a significant contribution in the multivariate model ($p \leq 0.05$). Coupling was eliminated from the multivariate model during the

Table 8.2: Results of multivariate analysis of sequence diagram LoD measures for the BHN project. The results show that SD_{msg} is not a significant predictor of defect density, after controlling for the effects of coupling and complexity.

Model	Unstdized. Coef.		Std. Coef.	t	Sig.	Correlations		
	B	Std. Error	Beta			Zero-order	Partial	Part
(Constant)	2.425	.275		8.823	.000			
SD_{msg}	1.077	.597	.243	1.803	.080	.319	.295	.241
Complexity	-.328	.081	-.545	-4.044	**.000**	-.579	-.570	-.540

Model Summary: $R^2 = 0.39; p \leq .001$.



(a) IPS Project

(b) BHN Project

Figure 8.2: The relationships between SD_{msg} and defect density in the IPS and BHN projects

backward elimination because of its little contribution to the model. Notice also that the b-value of SD_{msg} is positive, which suggests a positive correlation between SD_{msg} and defect density. If this result were statistically significant, it would be interesting to further assess a contradiction to the finding in the IPS project. Nevertheless, given the result is not statistically significant, further interpretations of it would be anecdotal. Figure 8.2 provides scatterplots visualizing the correlations between SD_{msg} and Defect density in the IPS and BHN projects.

An important lesson that we have learnt from analyzing multiple case studies is that generalizing results across case studies should not be taken lightly. Notwithstanding the similarity in various aspects such as software process method and configuration management system, differences between case studies are inevitable and may pose substantial impacts on the outcome of the respective case study—take developer experience as an example. Clearly, our study is never meant to quantitatively assess these differences as to perform a formal comparative study on the impact of using UML across projects. Such a study will require different approaches in terms of study design and the selection of software projects.

The fact that only one case study confirms the benefits of UML modeling can be considered as a limitation to the generalizability of this study. More specifically, this limitation sets some prerequisites in terms of project characteristics in which similar results are expected to emerge: 1) the UML model serves as a guide for the implementation—as opposed to mere documentation; 2) the UML model is created at certain levels of quality (e.g., in terms of level of detail and completeness); 3) developers conform strictly to the UML model; 4) testing is done thoroughly and defects found are registered systematically. While this list of characteristics is by no means complete, it serves as a fundamental checklist to identify software projects in which the results of this study would be applicable.

8.4 Recommendations

In Figure 8.1, we have shown, according to empirical evidence observed in our study, how the quality in UML models affects the quality of the final software product.

The figure shows that the use of UML increases system comprehension, which will subsequently reduce the introduction of defects and shorten the time to fix defects. In the end, these will translate to higher quality software and higher development productivity. Aside from that, the use of UML enables early quality predictions of the final software product. Not only can these predictions be used to anticipate or avoid defects before deployment, but they also can be used to prioritize testing effort on the most fault-prone software components. Again, these gains will increase both quality and productivity.

In order to achieve the quality and productivity improvements through the use of UML in software development, we recommend to consider the following:

- Target the level of quality in UML models according to their purposes. In particular, UML models that serve as guides for implementation will require higher quality level than those used only for communicating design decisions.
- If UML models are used for guiding the implementation, ensure that sufficient attention (e.g., level of detail and completeness) is given to complex and critical components/behaviors in the models.
- Consider using modeling guidelines such as naming conventions and enforcing the practice of maintaining model – code correspondence.
- Pay extra attention to behavior modeling such as sequence-, state-, and collaboration diagrams. Coding activities often rely heavily on such behavioral diagrams.
- Perform lightweight preventive and evaluative actions to assure the quality target for the UML model is achieved. For example by introducing design inspection and design checklist for quality assurance. Tool support to assess model quality would be a great help to automate the quality assurance process.
- Put UML models under a version control system to allow integrated and automated quality assurance of models.

8.5 Concluding Remarks

In various software engineering process methodologies (e.g., waterfall, iterative, v-model), the design phase exists to facilitate a smooth transition from high-level requirements and architecture to working software. As such, the role of software designs is to allow software designers and architects alike to express solutions or requirements in a more formal manner such that various analyses (e.g., consistency, correctness, modularity checks) of the software-to-be are possible before construction. Such analyses help clarify risks and uncertainties well in advance, and thus reduce costs incurred due to unnecessary rework.

The role of software designs is also crucial to communicate design decisions. This is particularly true for software projects whose team members are distributed across different locations (distributed teams). Constrained by various team backgrounds, such as domain knowledge and culture, projects with distributed teams can benefit from using common modeling language conventions to communicate design decisions.

Apart from the aforementioned role of software designs, it is also widely known that creating software designs is not a trivial task and it requires substantial investments. Hence, it is not surprising to see some emerging methodologies in software engineering that advocate devoting more effort in writing code and paying less attention on creating software designs.

Motivated by the above contexts, this study aimed to objectively assess the benefits of software modeling during software development. In particular, we focus on investigating the effects of modeling on the quality of software. Our attention during the investigation has been drawn primarily to UML (Unified Modeling Language)—the de facto industry standard for software modeling. Using multiple research strategies (i.e., survey, experiment, and case study) we have found convincing evidence about the benefits of using UML for improving the quality of software.

The findings of this study should encourage more research to develop methods and techniques for quality assurance of software models. For software engineering practice in general, the findings of this study confirm that software modeling does deliver added value to the quality of the final software product. Therefore, the question software engineers should ask is not whether they need or do not need modeling. Rather, the questions should be: when, where (which parts), and how much should they use modeling.

8.6 Future Research

We consider this study as an early step towards a comprehensive understanding of the role and benefits of modeling in software development. Replications of subsets of this study are obviously an immediate recommendation in order to enhance the generalizability of the results. Nevertheless, based on emerging questions during this study, we identify potentially interesting topics for future research:

- **The economics of modeling**
  Research in the area of economics of modeling should investigate the benefit of mod-

eling by taking into account the amount of investment put into modeling activities. This investment can then be compared with productivity and quality gains obtained in the construction, test, or maintenance phase. The major hurdle that kept us from investigating the issue was because we have no data about the cost of modeling. Such data is very difficult to obtain from ongoing projects, software repositories, or experiments. Researchers who have access to any of such data sources should definitely consider this topic an interesting research direction.

- **The exact form of 'model quality' - 'software quality' relationship**
  One of the findings in this study suggests that increasing the level of detail in UML models will reduce defect density of the software system. On the one hand, this finding gives us valuable input on how to use details in UML models. On the other hand, it also triggers another important question concerning the exact form of the functional relationship between model quality and software quality. It is logical to think that after a certain point increasing the quality of models would not result in a substantial quality improvement (diminishing returns). By understanding the exact form of the relationship between model quality and software quality we might be able to extrapolate some thresholds that can be used as stopping criteria for modeling.

- **Testing prioritization based on quality prediction**
  Defect prediction models provide risk assessments for software modules/classes to contain faults. In this study, we have come up with design metrics (as predictors) which can be collected in the design phase. An interesting future research would be to use the proposed metrics to perform predictions that can be used for prioritizing testing effort—that is, more modeling and testing activities should be devoted to software parts with high probability to contain faults. Furthermore, to assess the practicality of such approach, it would be particularly interesting to perform the study in running projects.