# How effective is UML modeling? An empirical perspective on costs and benefits

Chaudron, M.R.V.; Heijstek, W.; Nugroho, A.

Softw Syst Model (2012) 11:571–580
DOI 10.1007/s10270-012-0278-4

EXPERT'S VOICE

# How effective is UML modeling ?

## An empirical perspective on costs and benefits

**Michel R. V. Chaudron · Werner Heijstek · Ariadi Nugroho**

Received: 23 October 2011 / Revised: 23 July 2012 / Accepted: 27 July 2012 / Published online: 26 August 2012


**Abstract** Modeling has become a common practice in modern software engineering. Since the mid 1990s the Unified Modeling Language (UML) has become the de facto standard for modeling software systems. The UML is used in all phases of software development: ranging from the requirement phase to the maintenance phase. However, empirical evidence regarding the effectiveness of modeling in software development is few and far apart. This paper aims to synthesize empirical evidence regarding the effectiveness of modeling using UML in software development, with a special focus on the cost and benefits.

**Keywords** Unified Modeling Language · Costs and benefits · Quality · Productivity · Effectiveness

## 1 Introduction

Over the past decade modeling has become a common activity in software development projects. There is however, very little evidence about the effectiveness of using UML. As a result of this lack of scientific knowledge, debates continue about whether or not modeling helps to improve software development in practice.

Communicated by Prof. Jon Whittle and Prof. Gregor Engels.

M. R. V. Chaudron · W. Heijstek (✉)
Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
e-mail: heijstek@liacs.nl

A. Nugroho
Software Improvement Group, Amstelplein 1, 1096 HA Amsterdam, The Netherlands
e-mail: a.nugroho@sig.eu

In this paper, we present a selection of empirical evidence about the effectiveness of UML modeling in practice. We define effectiveness as the combination of positive (benefits) and negative (costs) effects on overall project productivity and quality. To this end, we also review what is known about the manner in which projects use UML. In particular, this paper will discuss:

- Empirical evidence about the effectiveness of UML modeling in software development, focusing on a costs and benefits perspective, and on industrial practice.
- Gaps identified in the research concerning effective UML modeling.

## 2 The practice of software modeling

The state of the practice of UML modeling has been surveyed by several papers since 2006: Grossman [10], Dobing [6], and Lange [19]. These papers survey the types of diagrams used, the industries in which modeling is used and try to relate this to other factors such as size of the project and managerial involvement in UML use. The main findings from these surveys are that three types of diagrams are most commonly used: use case diagrams, class diagrams and sequence diagrams. In embedded system projects, behavioural diagrams (sequence, state-charts and collaboration diagrams) are more common than in information system type of projects. A survey with a slightly different focus, yet very interesting in its findings, was performed by Forward and Lethbridge [8]. Their focus is on the perception of software professionals on the pro's and con's of modeling. In addition to these surveys, the authors of this paper have been visiting 20+ software development projects that use UML over the last 10 years. In the remainder of this paper, we present an empirical

perspective on the cost and benefits of UML modeling in industry based on these surveys and our industry visits. Moreover, we try to link cost and benefits to case studies and experience reports that have been published on this topic. Because we are aiming at a qualitative synthesis of evidence, we will combine findings regarding different approaches of UML modeling also including model-driven development (MDD) approaches which we define as approaches where code is automatically generated out of UML models. Hutchinson et al. [14] recently conducted a multi-method study of the state of the practice of MDD for which over 250 professionals were surveyed. The results of this study will be discussed later.

### 2.1 To model or not to model—who decides?

We describe how the decision whether or not to model is made. This section summarizes a discussion with 38 software professionals at a developers-community meeting at the university of Leiden in June 2012. These developers form a community for exchanging technical experiences and for general networking. Their experience ranges from 5 to 25 years of industrial experience in software development, following either a B.Sc. or M.Sc. education. These developers work both in small-to-medium sized enterprises and in large corporate IT departments. All 38 developers were familiar with the UML, a small minority (2 developers) was familiar with UML only as reader of diagrams, but had never created a diagram.

A discussion was held on who would decide whether to use modeling in a software development team/project. The following emerged: in small project teams (of at most 5 persons) the decision to model was either taken democratically[1] through voting, or championed by leading developers in the team. In small teams, the use of UML was not felt to be critical for the success of the project. Leading developers do consider fluency in, and use of UML as a sign of professionalism. Teams that are larger or more geographically dispersed are more inclined to decide in favour of modeling using UML. In larger and distributed teams, the perceived benefit of using UML is to have a shared common representation of the system to be built. The fact that such a commonly agreed representation is needed is supported by Weigert [32], who reports inconsistent interpretation and misunderstanding as drawbacks of less rigorous specification methods in comparison to model-based specifications. Also, for teams in larger organizations, there are sometimes corporate standards for software development which prescribe the use of UML. In this case, the decision to use modeling has been endorsed at the corporate level.

[1] This style of decision making in teams may well be very culturally dependent.

Staron [29] reports criteria used for adoption of model-driven development in Scandinavian companies that produce complex embedded systems. Adopting MDD has more impact on the entire development process than only using UML modeling. Adopting MDD puts more stringent requirements on the availability of professional tooling as well as on the model-centric orientation of the end-to-end development process; e.g. estimation and testing activities also need to be(come) model based.

Summarizing, the factors that lead teams to chose for using UML are: size of the team (and hence project), number of development locations involved and the presence of champion developers.

Once a decision to use UML for design modeling is made, there remains significantly different manners in which the UML is used. These differences in styles are the topic of the next section.

### 2.2 Styles of modeling

The costs and benefits of UML modeling depend on the manner in which UML is used. Therefore, we discuss in this section, the different styles of modeling in industrial projects.

#### *2.2.1 Models for analysis and understanding*

The most ‘loose’ style of modeling is ‘modeling for analysis and understanding’ or ‘modeling as a sketch’. Because this way of modeling is mostly done for own understanding, it is often sufficient to create sketches on a whiteboard. Sometimes these sketchy models find their way into persistent documentation as a digital picture of the drawing that was made on a whiteboard. The extra amount of effort required to meet the rigour required to (re)create the design in a UML modeling tool that insists on strict adherence to formal syntax is considered not adding value to the understanding. Therefore, this rigour is often not applied. The models created for this purpose are loosely based on the UML notation and typically no effort is made to make the models syntactically correct.

In this loose style of modeling, developers create an informal model of a system mainly for increasing their own understanding of the essential parts of a system. Modeling is a way of ordering and creating a structured abstraction of the real world. In this way, a model is a way of managing complexity—both mentally and of the problem domain. The creating of models of the system to be built, forces the designer to think hard about his system. While the same claim is being made by ‘formal methods’ advocates, one distinguishing feature of UML-based modeling is that it allows a varying degree of details across different parts of a model. Through developing an external representation, a developer can reduce the cognitive complexity of managing many details of the design in his mind.

If the representation and the method that are used for modeling are more rigorous, then they offer more opportunities for checking (validation and verification) the model. In this sense, both modeling, formally specifying and prototyping all share in their approach that they force the developers to think deeply about issues of a system's design before the actual system is built [3]. These approaches differ in their rigour and as a consequence they differ in the amount of effort that they require to be spent on less significant aspects of the system design to get a valid model.

Hutchinson et al. [14] report similar findings: 73.4 % of the people that have adopted MDD in their development process assert that models are used for understanding a problem at an abstract level.

Further benefits of increased understanding obtained via this style of modeling are also incurred by the more elaborate style of modeling that we discuss next.

#### *2.2.2 Models as a vehicle for communication*

Software is developed in teams. A representation of the design is a way of sharing information regarding the design amongst all team members. In this style of modeling, we see whiteboard sketches and also models created using generic drawing tools (such as PowerPoint and Visio) and sometimes using UML CASE tools. However, the models used for this purpose typically are not very detailed. The use of a well-defined notation helps in reducing the ambiguity in the representation, and hence in reducing misunderstanding within a team. In their study on MDD in an automotive company, Mellegard and Staron [22] cite an engineer's justification for using (use case) models: "without having these requirements for interfaces clearly and correctly stated, one would risk developing the function based on the wrong assumptions."

Even if the UML syntax may not be followed very precisely, the concepts of UML form the de facto language for discussing software designs. The findings of Hutchinson [14] support this finding: 73.7 % of the people that have adopted MDD in their development process assert that models are used for team communication. Moreover, developers from [14] find that using models in this way has a positive impact on system quality and productivity of development of the system. In their observational study, Cherubini [4] identify communication as the most common purpose of modeling.

A system's design mostly crystallizes during the initial 20 % (see, e.g. Fig. 4) of project time and evolves only a little during the remaining project time. As a result, we see that projects that employ this style of modeling invest effort in creating models in the early phases of a project and that the interest in keeping the models updated with changes decreases over time.

#### *2.2.3 Models as a blueprint for the implementation*

In this style, models are meant as the basis for programmers to create the implementation. All details that are considered informative for a programmer are included in a UML model. For this purpose the model typically includes use cases, class diagrams with methods and attributes, and sequence diagrams which list names of operations and sometimes types of parameters. Even for this style, still some design decisions are left open by the model. For this purpose of modeling, UML case tools are used. Such tools assist in creating syntactically correct diagrams. We found this style of modeling used in projects that develop software with teams at multiple geographic locations. In this context, the aforementioned argument is used: the rigour of UML is used to avoid misinterpretation when personal communication is scarce and difficult. Designers on one side of the ocean try to ensure that programmers on the other side of the ocean do not get important details wrong by explicitly including many design details in the model.

### 2.3 Models as program

When the implementation is generated from the model automatically (as is the case with model-driven development), then the UML model needs to include all details needed in the implementation code. The amount of information needed for this purpose can be so large that it becomes a threat for understandability and communication of the design. Fowler has phrased this issue as: "comprehensiveness is the enemy of comprehensibility".

A benefit of having models as programs is that these representations can be used as prototype for early feedback and for automated test generation. According to Hutchinson [14] 37.8 % of people that have adopted MDD, use models in testing and 41.7 % use model simulations (for verification and validation purposes).

### 2.4 Modeling trade-off

All approaches to modeling except modeling as program have to decide on the difficult question 'how much modeling is good enough?' This requires making a trade-off between the amount of effort that projects are willing to invest preventively in creating a high quality model, versus the extra effort or costs that are incurred as a result of the need to repair problems that arise due to imperfections in the model (so-called "non-conformance" cost). This trade-off is illustrated in Fig. 1. Typical types of imperfections in UML models are inconsistencies, incompleteness, and sometimes redundancy [20]. Through experiments [18] and case studies [24], we know that such imperfections lead to diverging interpretations of the model by different designers and to pro-

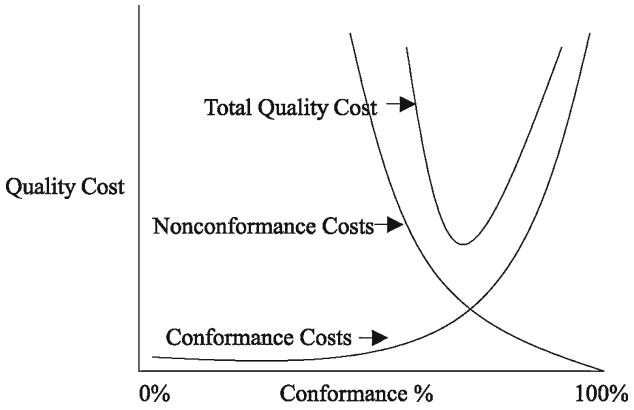


**Fig. 1** Cost of quality

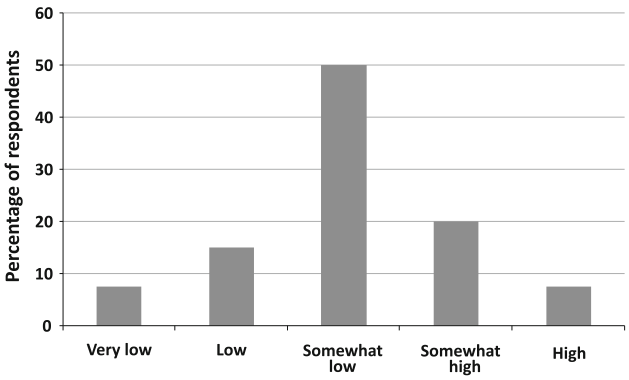


**Fig. 2** Developer opinion on adherence to OMG's UML standard

grammers, filling in the underspecified parts in the model in inconsistent ways in the implementation. For such deficiencies, it holds that repair at the downstream moment at which it is detected costs more than it would have cost to eliminate the imperfection in the model that caused the downstream problem as part of the modeling activity. However, not all imperfections in the model will cause downstream problems and therefore eliminating all imperfections from the model is considered impractical as it would require too much effort for design that may potentially change in downstream development.

Without further evidence, developers have no other option than to approach assessing cost of non-conformance from a stochastic perspective: what is the chance that the user of the model will make a mistake if this information is left out of the model and what will be the impact and cost of that? However, we know from a survey among professional developers [23] that they focus their modeling effort on complex and critical elements of the design. As a result, low risk and low complexity aspects of the system design get less attention in the model, and in this way developers economize on modeling.

These different styles of modeling found in industrial practice illustrate that different teams have different goals for their models. Consequently, the ideal level of quality for each of these approaches is different. However, currently there are no means to even approximate the question of how much modeling is good enough because there is too little understanding of the impact of lack of completeness of UML models on downstream development quality and productivity.

In our research, we collected a set of 40 software architecture design documents that are all based on the RUP-template. Architecture representations are at a high level of abstraction and often used for communication rather than implementation or code generation. We found that even within individual project the choice of rigour differs per diagram: 50 % of the diagrams can be considered as conforming to UML, and 50 % of the diagrams use other looser notation such as box-and-lines and/or includes domain-specific visualizations [12]. This suggests that designers balance the communication value of their representation with the rigour in following UML standards.

Next, we present some graphs based on our survey [23] that show the actual trade-offs that are made for modeling with UML in practice. Figure 2 shows the degree to which developers adhere to the OMG UML modeling standard. Overall, the adherence seems normally distributed from very loose to very strict with an average of loose. Furthermore, within the UML model of a system the level of detail is generally not uniformly applied. In a survey, we asked what motivated developers to add or leave out details in a model. The answers (Fig. 3) indicated that developers apply more details to complex and critical parts. In industrial practice, designs are moved into implementation stage while there are still large degree of incompleteness and inconsistency in a design. The developers of these projects currently feel that this is an adequate trade-off between effort in modeling and risk of problems in downstream development. To better support this practice of balancing modeling quality and downstream risk, methods for quality assurance are needed that also vary the level of quality they require of different parts of the system. In particular, more rigour must be exercised in critical parts and less rigour in less critical parts. This opens up the question how to distinguish critical parts in a system design from less critical parts of the design. As far as we know, this question has not been answered in the testing community where this question also arises in the context of ranking defects.

Answering the "How much modeling is good enough"—question requires the development of a notion of quality for UML models. The state of the are in the area of quality of UML modeling has recently been summarized in [9]. However, most of the research in quality of modeling focuses on syntactic issues of models. And while this has value, as witnessed by common use of code-quality checkers for source code, modelers are not so much interested in conforming to

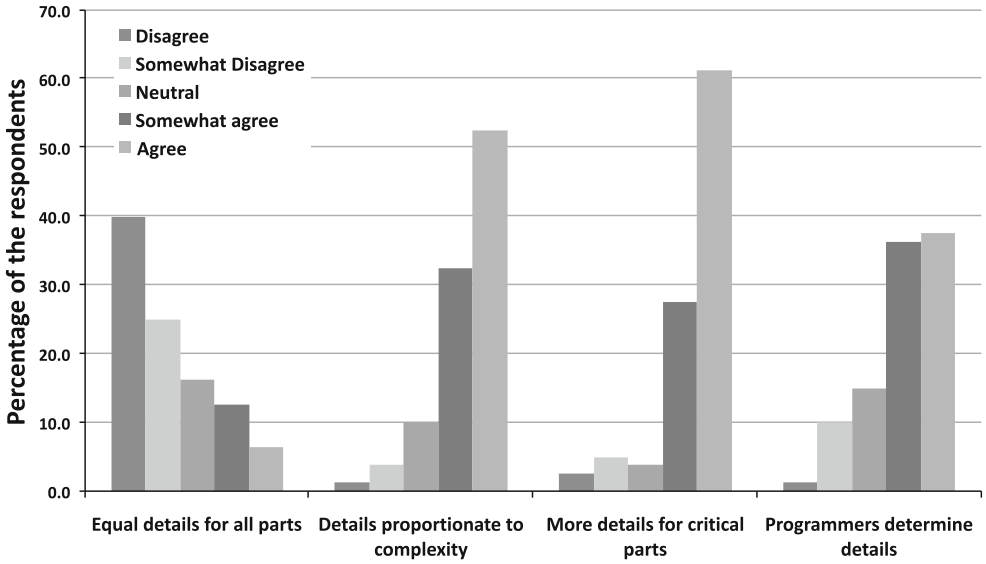


**Fig. 3** Respondent agreement over approaches in using detail in models

the precise syntax of UML or in inconsistencies in the model. Instead modelers are mostly interested in weak spots of the design. This triggers a research direction: we must develop better methods and tools for identifying flaws at the semantic level of designs.

## 3 Costs and benefits of modeling

In the literature concerning empirical studies into the use of UML, the cost of UML modeling remains a big question mark. Very few of the reported experiments show a clear picture regarding the cost of modeling. In this section, we try to gain insight into this matter by combining some studies that do state results that relate to costs and benefits of modeling.

### 3.1 Costs of modeling

One obvious parameter that could explain the cost of modeling is the size of the model. However, naively measuring size of a model is not a good indicator for the effort (and hence cost) of modeling. The evidence that we know so far is the following:

In an experiment, we offered 12 M.Sc. students a single set of requirements on two pages of A4-paper and allowed them 4 h to create a UML model for the system that meets the requirements. The smallest solution contained 7 classes, the largest solution contained 23 classes. No student felt that they needed to cut corners because of time limitations. These models were created within the same time span, but by developers with different skill levels.

In another experiment [17], participants were 106 M.Sc. students grouped into 35 teams. This time, all teams were asked to develop a model based on the same set of requirements, but now spread over a period of a semester (about 12 weeks). Here, the majority of the models were between 20 and 50 classes. Nugroho [25] comes to the conclusion that class size is too coarse a measure for characterizing the size of a model, and hence for predicting effort of modeling projects. He argues that additional factors like the following need to be taken into account: the level of abstraction used in modeling, the level of detail in modeling (are methods, attributes, labels, multiplicities etc. included or not) and last but not least: designer's capabilities (skills, knowledge and experience).

We have collected data about size of models and effort from a set of 14 industrial projects that employ modeling [19]. From these cases studies we found no statistically significant correlation between model-size (in terms of number of classes) and modeling effort. This may be explained by the fact that different projects employ different styles (and hence rigour) of modeling. In this dataset, a better predictor for the effort spent on modeling seems to be the number of members in the development team.

A comparison between the student experiments and the industrial case study is interesting. While students apparently manage to create a 20-class model in a 4-h session, industrial projects report a modeling effort in the order of weeks or months (i.e. a 10-fold increase) for creating models of the twice the number of classes.

In addition, in a study of a large case of model-driven development where over 90 % of code was generated from diagrams, we could not find any relation between the size of diagrams and the effort that was spent engineering them [11].

In a study of a large project, we collected data about the growth of the size of the models and the growth of the source

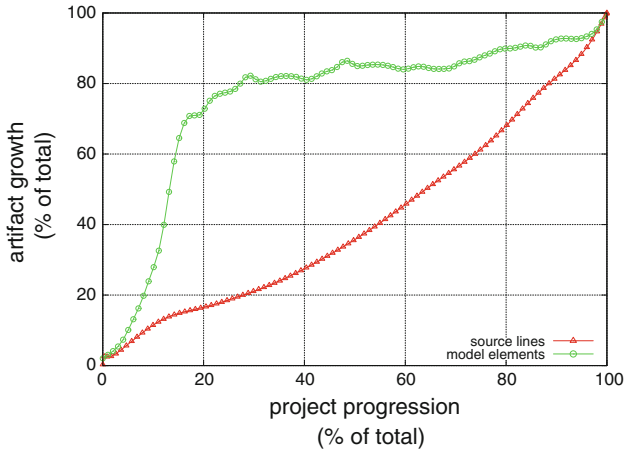


**Fig. 4** Model size versus code size across (normalized) project time

code. The size of the models was computed in a straightforward manner as the sum of the all modeling elements that appeared in the model. In this MDD case, code was generated out of the model, hence needed to be as 'complete' as any programming language implementation. This is typical for MDD projects that there is no variation in level of detail within a model. The graph in Fig. 4 shows the growth of the model in an MDD project and compares this to the average growth of the source code of three projects. Project duration has been normalized at 100 % and the sizes of the artifacts have been normalized at 100 % being equal to the size of the final artifact. The graph shows that in the first 20 % of the project time, 80 % of the model has been constructed. In the remaining 80 % of the time, the size of the model grows only very little. Given that the size does not change very much, it seems that in these last 80 % of the time the work on the model consists mostly of changes to it. The pattern for the growth of the code looks very different. Code seems to grow in an almost linear manner across the duration of the project. The graph can be interpreted as follows: the model approach starts with outlining the big picture and then goes on to flesh out the details. The coding approach seems to continue to add functionality across the entire project duration. Current views on iterative development favour the early delivery of value. This graph shows that this is very much compatible with the model-driven approach to software development.

In conclusion, we can discard the conjecture that capturing the models into a UML modeling tool is a time-consuming activity. A more likely explanation is that for larger systems, a majority of the time attributed to modeling is actually used for establishing consensus about the problem domain, about terminology and about the main design decisions across all stakeholders. This effort grows with team size and decreases with the skill level of the people on the team.

A recurring comment on the hurdles for modeling is the effort needed for keeping models and code synchronized e.g. [31]. This is also listed as the number 1 hurdle in [8]. In the same survey, the second and third most mentioned problems with modeling in software development are: models cannot be easily exchanged between tools, and modeling tools are 'heavyweight' (install, learn, configure, use). Whereas these impediments can be accounted as costs, they are better attributed to inadequate tooling. Arguments of this type should be categorized as hygiene factors. This terminology is borrowed from [13] and in our domain refers to the category of factors that is separate from positive motivation for the use of UML, but instead lead to dissatisfaction with the modeling activity in case of their absence.

Reviewing the evidence presented, we believe that the trade-off for using modeling is not so much driven by a financial trade-off, but by a lack of tool support.

### 3.2 Benefits of modeling

In this section, we present empirical evidence regarding the benefits of UML modeling.

The fact that the industrial engineers did not worry about the large degrees of incompleteness and inconsistencies in their UML models, begged the question as to whether these characteristics do or do not lead to problems in downstream development. We studied this through an experiment [18]. In this experiment, we showed UML models with and without defects to subjects that consisted of both students (about 75 % of our population) and professionals (about 25 % of our population). We asked our subjects to answer questions that tested their understanding (or: interpretation) of the models. The outcome of the experiment showed that the presence of defects in UML models led to more variety in the interpretation of the models. Given that one of the key purposes of UML models is to communicate one particular design to all team members, this is already a serious indication that consistency and completeness in modeling should be part of quality assurance practises. Our industrial partners replied that the implementation process was robust to these inconsistency issues and that communication between developers would filter out any misinterpretation. Subsequently, we studied this claim by studying the effects of the quality of the model on the quality of the implementation. To this end, we studied a project repository that included (1) the UML models that were the basis of the implementation, (2) the final source code and (3) the defects found in the testing of the source code [24]. This study showed that the defect density of classes in the implementation was negatively correlated with the quality of the sequence diagrams in which these classes appeared. Hence a lower quality of UML models correlates with a lower quality of the final source code. At first, we were puzzled that we did not find a similar correlation for the quality of class diagram and the implementation. It is possible that this was just a random effect of this particular case study. We did, however, find

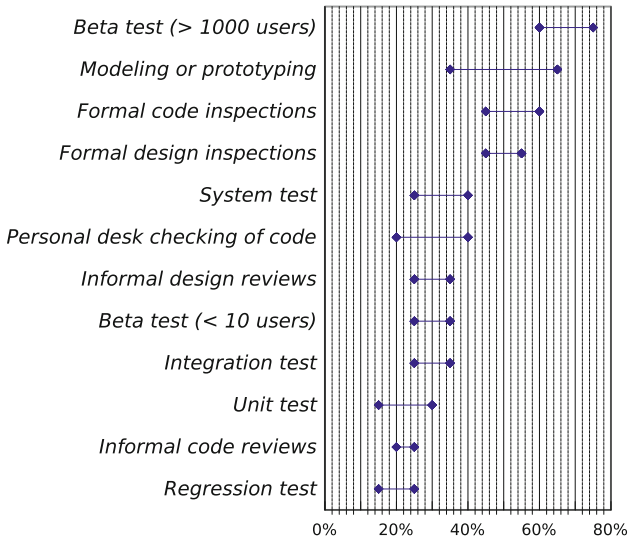


**Fig. 5** Effectiveness of defect detection methods (summarized by McConnell [21], based upon work by Jones [15,16] and Shull [28])

a reasonable explanation for this observation: Defect repositories contain defects that are found through testing. Mostly these are results of faults that cause an incorrect value or a failure of the software. The nature of these defects is that they relate to the behaviour of the system. This is the link to sequence diagrams: these describe the behaviour. Nevertheless, we hypothesize that the quality of the class diagram could correlate with a better structure (e.g. in terms of better modularity) of a design. Currently, we only have the consent of developers with this conjecture [23] and are performing ongoing research in this direction.

From a quality assurance perspective, we can consider modeling as a means of identifying defects. McConnel [21] has compiled a graph which shows that modeling is an effective method for defect detection (Fig. 5).

In his seminal book on software engineering economics, Boehm [2] reports on the relation between the stage of a project (starting, middle, finishing) and the cost of repairing defects. The relation is such that the cost of repairing increases exponentially with the progress of a project. More precisely, this principle is nowadays formulated as that the cost of repairing a defect grows exponentially with the lifetime of a defect. From this perspective, it makes economic sense to start removing defects as early in the development as possible. Given that models are constructed early in development, they can be for reviewing and challenging a design rather than waiting until the implementation is ready and find defects through unit- and integration testing.

Complementary to the support for modeling implied by these quantitative studies, there are also more subjective pieces of evidence—obtained through surveys and interviews—in which designers and programmers state that they feel that the use of modeling aids in achieving a common understanding of a design within an team and that modeling helps as an aid in communication. These benefits ultimately lead to a positive effect on productivity and quality [23]. However, the impact of modeling on improved communication is very difficult to study. A noteworthy study in this direction is reported by Pareto [26]. Additional evidence that supports the importance of good documentation comes from an interesting corner: the documentation-averse agile software development community: a recent study amongst agile development team showed that they would prefer to have more documentation than they normally produce [30].

Hutchinson [14] specifically addressed the perceived impact of MDD activities on productivity and maintainability. They found that the largest impact was not code generation or meta-model reuse, but "the use of models for understanding a problem at an abstract level". The second greatest impact was thought to be "Use of models for team communication".

The fact that the use of models forces developers to think about these issues early in the project is considered to be economical in the sense that it prevents repair work that is typically more expensive the later the issues are found in the development [2]. In an early study that compares prototyping and specifying [3], Boehm reports comparable benefits for both approaches and explains this effect as the result of the fact that both approaches force early, yet thorough, thinking about design.

The evidence that we discussed is synthesized in Fig. 6, which shows how modeling leads to beneficial effects for software development projects. The complementary Table 1 lists from left to right along its columns the different styles of modeling in increasing order of level of rigour. The rows "model quality/rigour" and Archetypical approach characterize these different modeling styles. The next rows list an assessment on an ordinal scale about the relative costs and benefits of the corresponding modeling style. The assessment should be interpreted as relative to the other styles of modeling. Table 1 shows that modeling as a Blueprint for implementation and modeling for code generation have a relatively higher initial investment in creation of the model, but they also have higher benefits. In practice, the long term investment vision for IT development is influential in companies' decisions on which style of modeling they chose.

## 4 Related work

A large portion of the empirical research in modeling focuses on understandability by comparing different styles of UML modeling from the perspective of developers. More attention is needed for in-vivo studies of modeling where the creation, use and maintenance of UML models are studied. From our experience, the best level at which to per-

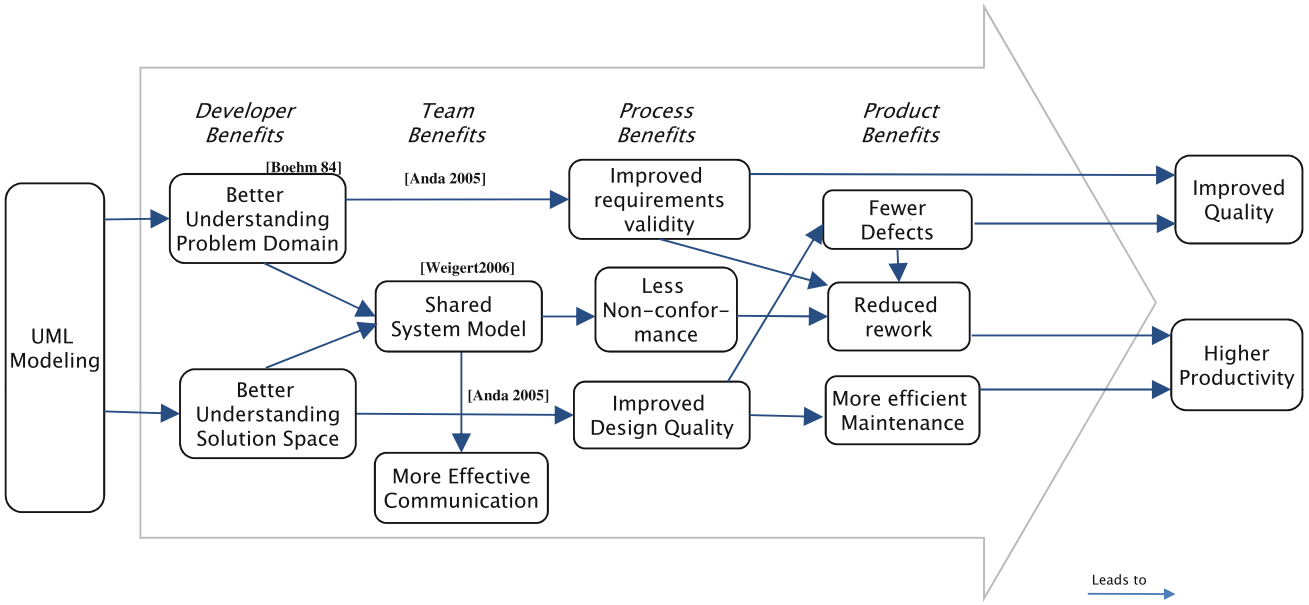


**Fig. 6** An overview of the benefits of UML modeling

form experiments is at the level of tasks, rather than projects or phases or teams. Anda [1] reports experiences from the introduction of UML modeling in a large project in Norway: *developers experienced improvements with traceability from requirements to code, design of the code, and development of test cases as well as in communication and documentation*. A study by Arisholm [7] evaluates the effect of availability of documentation on the efficiency and quality of performing maintenance tasks on source code. Their main findings were that *"[t]he quantitative results show that UML did not have a significant impact on the time that it took to perform the change tasks. […] However, in terms of the functional correctness of the changes, UML had a practically and significantly positive impact."* Given the theory of increased impact of early decisions in software projects, it is likely that the effect of using UML during maintenance on productivity is smaller than the effect of improved modularity that may result from modeling during the design stage.

Several case studies about use or adoption of UML and MDD have been published. Weigert and Weill [32] report their positive experiences about use of MDD at Motorola. Staron [29] reports a critical attitude, especially towards maturity of tooling, in a study of two cases (ABB and Ericsson).

## 5 Conclusion and future directions

Modeling is a common practice in software development and will become more common with the adoption of model-driven technology. The industrial practice of software modeling using UML is very diverse. Over the last decade, very little evidence has been obtained regarding the (in)effectiveness of modeling, especially quantitative data are very scarce. Some data have been published based on industrial case studies. The quantitative aspects of this data is very diverse. This is to be expected given the large diversity of context of the case studies. Yet, from a qualitative angle, there seems to be agreement on many findings (both positive and negative). Especially for larger and distributed project, UML modeling is believed to contribute to shared understanding of the system and more effective communication. This in turn avoids expensive rework that occurs in projects that skip modeling. Another important benefit is that early modeling help validation: ensuring that the right system is being built. Moreover, we have shown that the actual inputting of a model into a tool is not a time-consuming task. The actual effort is more likely spent on designing and communicating and agreeing on a design.

The evidence reviewed in this paper suggests that decision whether or not to modeling in a software project is not so much driven by cost factors. Instead, this decision is strongly influenced by a hygiene factor: inadequate tooling.

### 5.1 Future directions

The focus of this paper is empirical evidence about effectiveness of modeling. An important gap in the knowledge about using UML is related to the impact of poor quality UML models on downstream development quality and productivity. The research methods needed for studying this problem are difficult to design and perform. A promising approach is presented by Premraj [27] who trace back defects found in

**Table 1** Summary of costs and benefits for different styles of modeling

| Model quality/rigour archetypical approach | Modeling for analysis and understanding (very) low whiteboard | Modeling as vehicle for communication somewhat low drawing tool | Models as Blueprint for the implementation somewhat high modeling tool | Models for code generation high modeling tool and code generator |
|---|---|---|---|---|
| Model creation effort | + | A | - | - - |
| Early validation ⇒ improved requirements validity | + | + | ++ | ++ |
| Early verification ⇒ improved requirements consistency | - - | - | + | ++ |
| Shared understanding and improved communication | + | + | ++ | A |
| Maintaining design and implementation synchronized | Not applicable | - | - - | ++ |
| Code quality | No data | No data | + | ++ |
| Improved design ⇒ less rework | A | + | ++ | No data |
| Improved design ⇒ more efficient maintenance | - | A | ++ | ++ |

*A* average, - - much worse than average, - worse than average, + better than average, ++ much better than average

the project repositories to architecture issues. Furthermore, in such studies attention must be paid to distinguish between design activities and modeling activities.

From the studies discussed in this paper, we can also learn about needs of practitioners that are not fulfilled by current modeling tools. We mention some of these next.

For developers a practical problem that limits the adoption of UML is the insufficient integration of UML modeling tools with programming tools. A particularly bothersome tasks for which tool support seems feasible, but is currently not adequately supported, is the automated updating of UML models following progression of source code. In [29] Staron reports a consistent finding: tools for different activities in the software development lifecycle are not well integrated and not sufficiently model-centric. Following Dekel [5], software development tooling must acknowledge that modeling is not an isolated activity, but that it is integrated with sketching and explaining. Hence, software design tooling must integrate sketching, modeling and textual explanations.

## References


1. Anda, B., Hansen, K., Gullesen, I., Thorsen, H.K.: Experiences from introducing uml-based development in a large safety-critical project. Empir. Softw. Eng. **11**(4), 555–581 (2006)
2. Boehm, B.W.: Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ (1981)
3. Boehm, B.W., Gray, T.E., Seewaldt, T.: Prototyping vs. specifying: a multi-project experiment. In: Proceedings of the 7th international conference on Software engineering, pp. 473–484. IEEE Press, Piscataway (1984)
4. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J.: Let's go to the whiteboard: how and why software developers use drawings. In: Proceedings of the SIGCHI conference on human factors in, computing systems, pp. 557–566 (2007)
5. Dekel, U., Herbsleb, J.D.: Notation and representation in collaborative object-oriented design: an observational study. In: Proceedings of the 22nd annual ACM SIGPLAN conference on object-oriented programming systems and applications, pp. 261–280 (2007)
6. Dobing, B., Parsons, J.: How UML is used. Commun. ACM **49**(5), 109–113 (2006)
7. Dzidek, W.J., Arisholm, E., Briand, L.C.: A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Trans. Softw. Eng. **34**(3), 407–432 (2008)
8. Forward, A., Lethbridge, T.: Perceptions of software modeling: a survey of software practitioners. Technical Report TR-2008-07, School of Information Technology and Engineering, University of Ottawa, 800 King Edward Ave. Ottawa, Ontario, Canada K1N 6N5 (2008)
9. Genero, M., Fernández-Sáez, A.M., Nelson, H.J., Poels, G., Piattini, M.: Research review: a systematic literature review on the quality of UML models. J. Database Manag. **22**(3), 46–70 (2011)
10. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML make the grade? insights from the software development community. Inf. Softw. Technol. **47**(6), 383–397 (2005)
11. Heijstek, W., Chaudron, M.R.V.: Empirical investigations of model size, complexity and effort in large scale, distributed model driven development processes—a case study. In: Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2009) Patras, Greece (2009)
12. Heijstek, W., Chaudron, M.R.V.: On the use of UML diagrams in industrial software architecture documents. Technical Report TR2011-02, Leiden Institute of Advanced Computer Science, Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands (2011)
13. Herzberg, F.: One more time: how do you motivate employees? Harv. Bus. Rev. **46**(1), 53–62 (1968)
14. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of mde in industry. In: Proceeding of the

33rd international conference on Software engineering, pp. 471–480. ACM Press, London (2011)
15. Jones, C.: Software defect-removal efficiency. Computer **29**(4), 94–95 (1996)
16. Jones, C.: Programming Productivity. McGraw-Hill, New York (1986)
17. Lange, C.F.J., Bois, B.D., Chaudron, M.R.V., Demeyer, S.: An experimental investigation of UML modeling conventions. In: Proceedings of the 9th international conference on Model Driven Engineering Languages and Systems, pp. 27–41 (2006)
18. Lange, C.F.J., Chaudron, M.R.V.: Effects of defects in UML models: an experimental investigation. In: Osterweil L.J., Rombach H.D., Soffa M.L., (eds.) Proceedings of the 28th international conference on Software engineering. pp. 401–411. ACM Press, London (2006)
19. Lange, C.F.J., Chaudron, Michel R.V., Muskens, J.: In practice: UML software architecture and design description. IEEE Softw. **23**(2), 40–46 (2006)
20. Lange, C.F.J., Chaudron, M.R.V., Muskens, J., Somers, L.J., Dortmans, H.M.: An empirical investigation in quantifying inconsistency and incompleteness of UML designs. In: Proceedings of the Workshop on Consistency Problems in UML-based Software Development (2003)
21. McConnell, S.: Code Complete. Microsoft Press, Redmond (2004)
22. Mellegård, N., Staron, M.: Characterizing model usage in embedded software engineering: a case study. In: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume. pp. 245–252 (2010)
23. Nugroho, A., Chaudron, M.R.V.: A survey into the rigor of UML use and its perceived impact on quality and productivity. In: ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pp. 90–99. ACM Press, New York (2008)
24. Nugroho, A., Flaton, B., Chaudron, M.R.V.: Empirical analysis of the relation between level of detail in UML models and defect density. In: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems, pp. 600–614. Springer, Berlin (2008)
25. Nugroho, A., Lange, C.F.J.: On the relation between class-count and modeling effort. In: Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, vol. 5002/2008, pp. 93–104. Springer, Berlin (2007)
26. Pareto, L., Eriksson, P., Ehnebom, S.: Architectural descriptions as boundary objects in system and design work. In: Proceedings of the 13th international conference on Model driven, engineering, pp. 406–419 (2010)
27. Premraj, R., Nauta, G., Tang, A., van Vliet, H.: The boomeranged software architect. In: Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, WICSA '11, pp. 73–82. IEEE Computer Society, USA (2011)
28. Shull, F., Basili, V., Boehm, B., Brown, A. W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M.: What we have learned about fighting defects. In: Proceedings of the Eighth IEEE Symposium on Software Metrics, pp. 249–258. IEEE Press, USA (2002)
29. Staron, M.: Adopting model driven software development in industry—a case study at two companies. In: MoDELS, pp. 57–72 (2006)
30. Stettina, C.J., Heijstek, W.: Necessary and neglected? an empirical study of internal documentation in agile software development teams. In: Proceedings of the 29th ACM International Conference on Design of Communication (SIGDOC 2011), Pisa, Italy, (October 2011)
31. Thörn, C., Gustafsson, T.: Uptake of modeling practices in SME's. In: Proceedings of the ICSE workshop on Modeling in Software Engineering (MiSE), ACM Press, New York (2008)
32. Weigert, T., Weill, F.: Practical experiences in using model-driven engineering to develop trustworthy computing systems. In: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy, Computing (2006)

## Author Biographies

**Michel R. V. Chaudron** is an Associate Professor at Leiden University, the Netherlands. Prior to this [he] has worked at the TU Eindhoven. His research interests are centered around software architecure & design and software modeling, with a special focus on effectiveness of modeling in practice. To this end he favours empirical studies and data collection in industrial settings. Michel Chaudron is a steering committee member of the Euromicro SEAA conference and is organiser of workshops on Quality of Modeling and Empirical Studies on Modeling at the MODELS conference. Contact him via: mrvchaudron@gmail.com.

**Werner Heijstek** is a Ph.D. candidate at the Leiden Institute for Advanced Computer Science (LIACS) at Leiden University. His doctoral dissertation (which is accepted for defence) deals with the challenges associated with representation, dissemination and coordination of software architecture design in the context of global software development. It particularly addresses model-centric development and explores the impact of model-driven development (MDD) in a distributed development setting. From October 2012, Werner Heijstek will work as a consultant at the Software Improvement Group in Amsterdam. Contact him at w.heijstek@gmail.com.

**Ariadi Nugroho** is researcher at the Software Improvement Group, the Netherlands. His research interests include software quality, software economics, software architecture and model-driven software development. Ariadi received his Ph.D. in computer science from Leiden University. His Ph.D. research focused on quality assurance in model-driven software development, via empirical analyses of software models' metrics. One of the results of Ariadi's research was the definition and validation of a new suite of metrics for UML models. Contact him at a.nugroho@sig.eu.