

In Practice: UML Software Architecture and Design Description

Christian F.J. Lange and Michel R.V. Chaudron, *Eindhoven University of Technology*
Johan Muskens, *Philips Research*

A user survey and industry case study analyses reveal common problems with UML models and some techniques for controlling their quality.

Since its introduction in 1997, the Unified Modeling Language has attracted many organizations and practitioners. UML is now the de facto modeling language for software development. Several features account for its popularity: it's a standardized notation, rich in expressivity—UML 2.0 provides 13 diagram types that enable modeling several different views and abstraction levels. Furthermore, UML supports domain-specific extensions using stereotypes and tagged values.

Finally, several case tools integrate UML modeling with other tasks such as generating code and reverse-engineering models from code.

In most projects, UML models are the first artifacts to systematically represent a software architecture.¹ They're subsequently modified and refined in the development process. Their importance has increased with the advent of the model-driven architecture methodology (www.omg.org/mda). However, little is known about the way UML is actually used in projects and the pitfalls of UML-based development.

To find out how practitioners use UML, we invited software architects to complete a Web-based anonymous questionnaire.² We also analyzed UML models in 14 industry case studies and compared the analytical results with the practitioner reports. Our study focused on UML use and model quality in actual projects rather than on its adequacy as a notation or language.

Practitioner reflections on UML use

Over a two-month period, 80 architects participated. Background questions revealed the following major responsibilities among respondents:

- analysis (66 percent),
- design (66 percent),
- specification (61 percent), and
- programming (52 percent).

The respondents came from different application domains. Most worked in information systems (61 percent); 28 percent worked in embedded systems, and a few worked in tool and operating systems development. Sixty percent worked in projects of more than five person-years.

Use of architectural views

We investigated UML's popularity for de-

scribing the different architectural views suggested by Philippe Kruchten in his “4 + 1 View Model.”³ However, we adopted the Rational Rose case tool terminology for the views because it’s well-known among most practitioners. Figure 1 shows survey results regarding respondents’ use of UML in use cases, logical views, component views, deployment, and scenarios.

Adherence to UML specification

The UML specification (www.uml.org) defines the language notation, syntax, and (informal) semantics. Figure 2 shows how strictly practitioners think they follow this specification. Based on these self-assessments, adherence to the specification is rather loose. This might be a result of UML’s lack of formal semantics and large degree of freedom in its application. On the other hand, it might be that informal use is “good enough” for many practitioners’ purposes.

Stopping criteria for modeling

We were interested in the criteria that practitioners apply to determine when modeling activities can end. The most prominent criterion was completeness, which 52.5 percent of the practitioners selected. However, no objective criteria exist to verify that a model is complete or that it satisfies some tangible notion of quality. The second-most prominent criterion was passing a review or an inspection, which 33.8 percent of the practitioners selected.

When schedule or a deadline becomes the criterion for switching from modeling to implementation, it’s usually a poor substitute for a systematic review and a negative indicator for product quality. Nevertheless, 32.8 percent of the practitioners mentioned deadline as a stopping criterion—an alarmingly high percentage. Some 17.5 percent indicated the amount of effort spent as their stopping criterion.

In correlating respondent demographics with the question of stopping criteria, we found that the criteria varied with different project sizes. Most striking was the comparison between deadline and completeness. Figure 3 shows that deadline is more often a stopping criterion in smaller projects, whereas all projects larger than 99 person-years reported completeness as their stopping criterion. This is probably because increased complexity in both the product and the project likewise increases the need for a systematic approach.

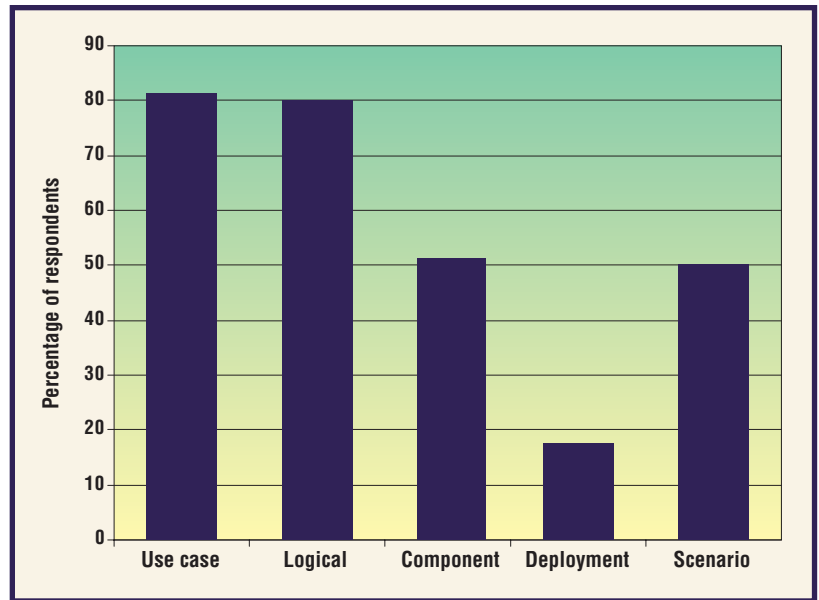


Figure 1. Survey respondents’ use of UML for architectural views.

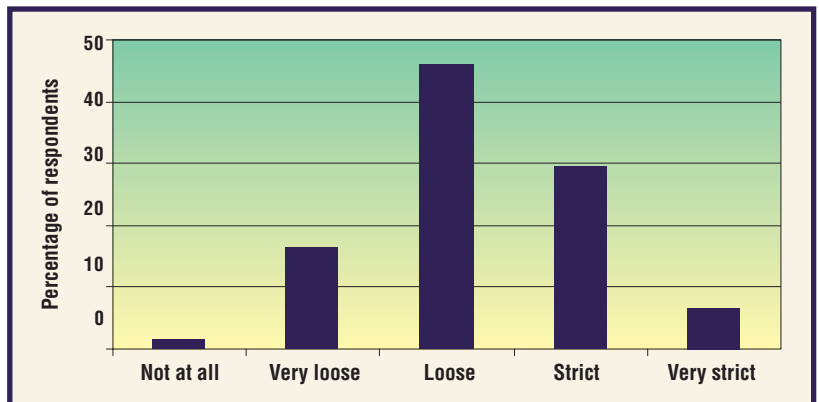


Figure 2. Survey respondents’ assessment of their adherence to the UML standard.

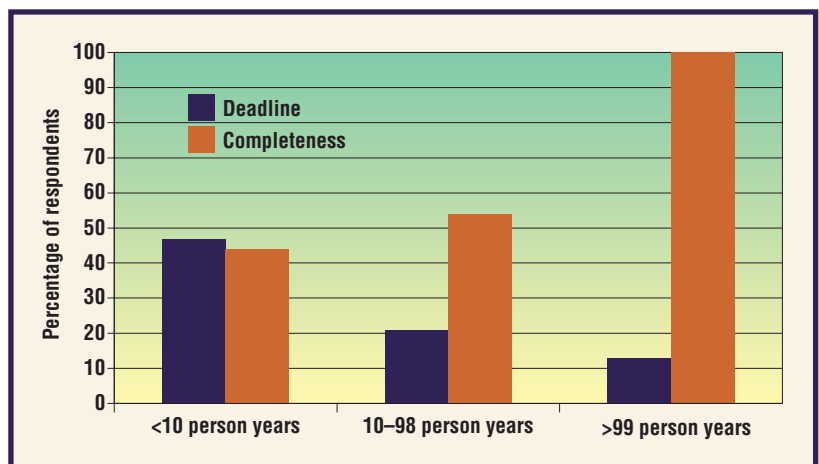


Figure 3. Deadline vs. completeness as stopping criteria for different project sizes.

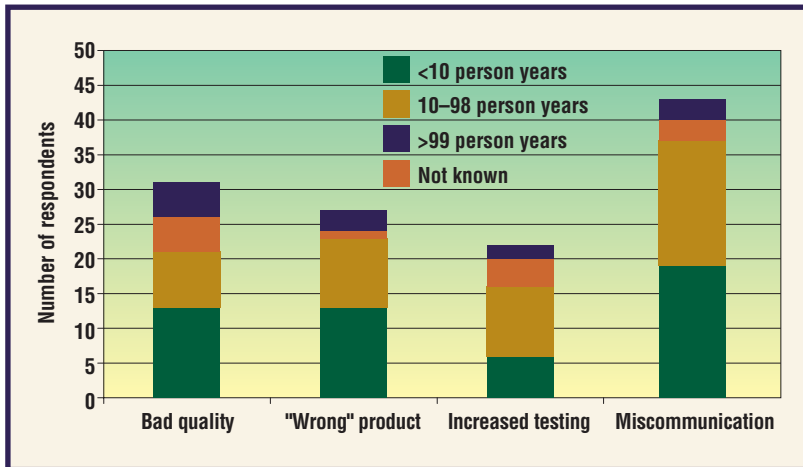


Figure 4. Problems encountered due to incomplete models correlated to respondent demographic data regarding project size.

Problems with incomplete models

Given that the practitioners report completeness to be the primary criterion for deciding to stop modeling, we investigated the effects of moving to the next project phase without a complete model.

Figure 4 shows four problems respondents encountered as a result of incomplete models. More than half reported miscommunication between project members and stakeholders when the project moved forward with incomplete models. They also mentioned poor quality of the implemented product; a “wrong” product delivered, in the sense of not matching the stated requirements; and high amount of testing effort.

Problems with UML descriptions

In both the surveys and additional interviews, architects indicated problems they encountered that are inherent in using a multidocument notation such as UML. On the basis of their responses, we identified four main problem classes:

- *Scattered information.* Design choices are scattered over multiple views. For instance, some dependencies might show up in the logical view, while others appear in the process view.
- *Incompleteness.* Many projects knowingly complete only a subset of the architectural views. The architects focus on what they think is important.
- *Disproportion.* Architects might work out more details for system parts that they consider to be more complex. If increased detail consistently indicated increased criticality or complexity, implementers could use this information in allocating testing resources.

- *Inconsistency.* UML-based software development is inevitably inconsistent.⁴ Industrial systems are typically developed by teams. Different teams can have different understandings of the system as well as different modeling styles, and this can lead to inconsistent models. Although some UML tools provide basic checks on consistency between diagrams, the scope of these checks is limited and leaves much room for inconsistencies to remain between views.

Incompleteness, disproportion, and inconsistency arise much more in software architecture models than in source code. This is because source code includes formal criteria for the form (grammar) and tools for checking these criteria (compilers).

Other problem classes include the following:

- *Diagram quality.* UML lets architects represent one design in different ways. For instance, they can decompose a diagram that contains too many elements into several smaller diagrams. Although different ways of organizing diagrams don’t change the actual design, they can influence how easy the model is to understand and how it gets interpreted.
- *Informal use.* Architects sometimes use UML in a very sketchy manner. For example, to obtain a better understanding of a system or to explain the architecture, they might use generic drawing software or even just make sketches on paper. These diagrams deviate from official UML syntax, making their meaning ambiguous.
- *Lack of modeling conventions.* Our case studies show that engineers use UML according to individual habits. These habits might include layout conventions, commenting, visibility of methods and operations, and consistency between diagrams. In programming, coding standards belong to the standard repertoire of quality assurance techniques.⁵ Similarly, “modeling standards” in the architecting phase would help in establishing more uniform UML usage.

Different uses of UML arise naturally as the design decisions and detail increase in both quantity and complexity throughout the design process. A sketch might be sufficient to capture

Table 1**Case study characteristics**

Case study	No. of classes	No. of person-years spent on modeling	No. of team members	Life stage of the model	Purpose of modeling	CMM level (estimated)
A	734	15	5	Final	Implementation	1
B	168	20	20	Unknown	Unknown	2–3
C	108	20	10	Unknown	Unknown	2–3
D	716	Unknown	Unknown	Unknown	Unknown	1
E	443	10	10	Final	Unknown	1
F	4	1	3	Final	Unknown	1
G	75	10	Unknown	Unknown	Unknown	1
H	478	Unknown	Unknown	Unknown	Unknown	1
I	705	12	6	Development	Implementation	1
J	51	12	6	Development	Analysis	1
K	14	1–5	2	Inception	Analysis	2
L	46	1–5	2	Final	Implementation	2
M	73	0–5	1	Final	Tutorial, abstraction of real world	1
N	359	5	5	Semifinal	Implementation	1–2

initial ideas, whereas significant detail goes into a system architecture that implementers will use to generate source code. Martin Fowler distinguishes three types of UML usage: as a sketch, as a blueprint, and as a programming language.⁶

The generality and freedom that enable UML to cater to this wide range of purposes are also the source of its weakness. UML has no formal semantics. This poses a problem when different people use a UML model; and because one of UML's main purposes is to communicate about a design, different ways of using UML are potential causes of communication problems.

For example, consider an architect who specifies only the most important classes, methods, and attributes for a model—omitting auxiliary classes that he or she thinks are straightforward. Colleagues working in a different office or country must use the architect's model to implement the system. If they assume the model describes every detail, they will misinterpret it.

Despite these problems, the large community of UML users is evidence overall of its usefulness.

Defects in industrial UML models

In addition to the subjective impression obtained via the survey, we performed objective measurements about the quality of industrial UML models. To this end, we implemented

a tool called MetricView (www.win.tue.nl/empanada/metricview) that checked the models for defects, disproportions, and risks of misunderstanding. We applied the tool to several industrial case studies of different sizes from various organizations and application domains (see table 1). The cases use a variety of UML CASE tools.

Defects found in the case studies

The case studies revealed several violations of UML modeling rules. While we don't expect all these rules to hold on any project, we counted model inconsistencies and incomplete spots in our measures of design quality and project progress.

Methods that are not called in sequence diagrams. To depict class interactions, the public methods that a class provides in a class diagram must be called in sequence diagrams. Methods that aren't called might not be used in interactions; alternatively, they might not be sufficiently described in terms of their interactive functionality.

Between 40 and 78 percent of the methods in the models we analyzed were not called in a sequence diagram.

Classes not occurring in sequence diagrams. If a design contains a class that doesn't occur as a

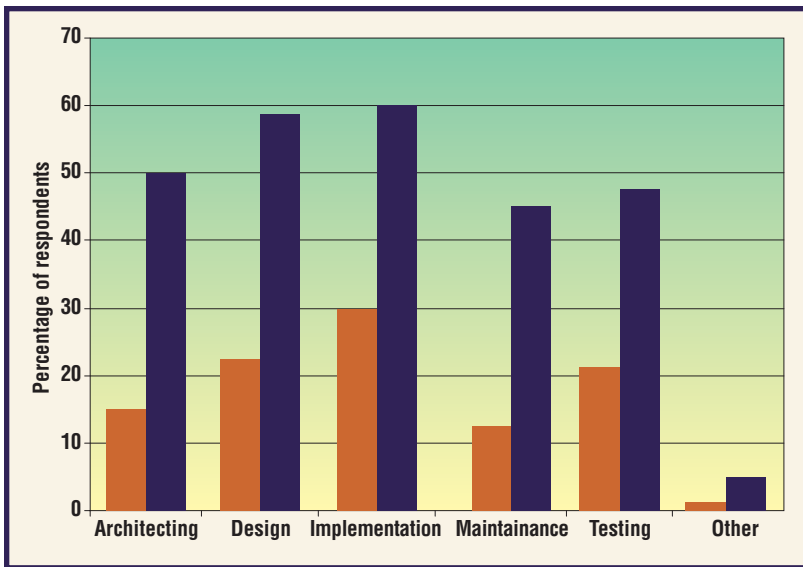


Figure 5. Responses to survey questions regarding the use of UML metrics: current use (brown) and desired use (dark blue).

sequence diagram object, either the class is redundant (assuming the sequence diagrams describe the entire functionality) or the interaction of the design's classes is not completely described using sequence diagrams.

In the models we analyzed, between 35 and 61 percent of the classes didn't occur as objects.

Objects without names. Each object in a sequence diagram must have a name. Named objects are more expressive and understandable than unnamed ones. Naming objects is essential, especially when several objects of the same type occur in one sequence diagram.

In our case studies, we found between 25 and 92 percent of all objects lacked names.

Messages not corresponding to methods. Each message that an object in a sequence diagram receives must correspond to a method in the object's class interface; otherwise the meaning of the message is unclear.

Between 8 and 59 percent of the messages in the analyzed models didn't correspond to methods.

Classes without methods. A class without a method violates the object-oriented paradigm, particularly the concept of encapsulation. A class without methods can't interact with other classes and is therefore incomplete. To make a class complete, the designer must define a class's methods and describe its interactions in a sequence diagram. Only in early modeling phases can you create classes without defining methods.

In all case studies beyond the analysis stage, between 20 and 60 percent of all classes lacked methods.

Effects of UML model defects

In spite of help from CASE tools, defects are common in software development. But what are the effects of UML model defects? Are they eventually detected, when someone uses the model as a specification for implementation? If not, does a defect really cause different readers to interpret the model in different ways?

We conducted a controlled experiment with 110 students and 48 practitioners to answer these questions.⁷ The results showed that defects often remain undetected, even if the model is read attentively for implementing the system. For example, 61 percent of readers didn't detect a sequence diagram message that lacked a corresponding method in the class diagram. The results are worse for an object that lacks a class specification: 82 percent didn't detect this defect.

These low detection rates raise the question of whether defects increase the risk for different interpretations. The experimental results showed varying risks for misinterpretation along a scale of 0 to 1, where 0 represents the widest spread in interpretations and 1 represents total agreement on one interpretation. For example, a use case that isn't described by a sequence diagram has a value of 0.44, representing a high risk for misinterpretation.⁷

Opportunities for improving UML usage

After observing these defects, we saw several opportunities for preventing and removing them.

Defect checking and feedback

Current tools provide only limited support for defect checking. Even though UML's informal nature doesn't support formal checks, the tools should be extended to automatically detect defects.

UML profiles have been used to define specific architectural styles and patterns—for instance, client-server patterns for use in telecommunications systems.⁸ The pattern defines which building blocks the model developer can use and what types of relations can exist between the blocks. The model developer can use these patterns constructively to guide the design. During development, tools can also verify that architecture, design, and implementation conform to this pattern.

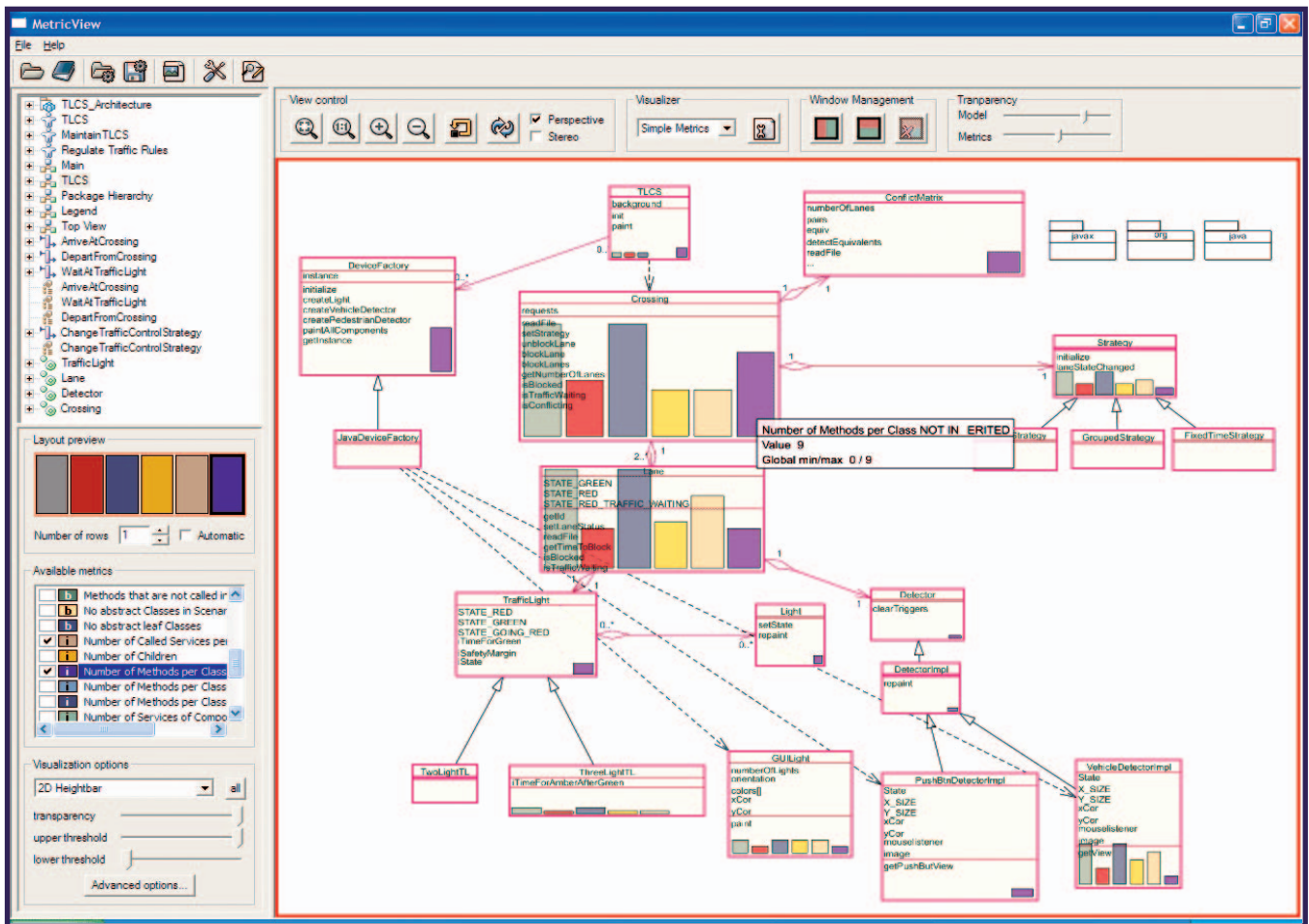


Figure 6. Screenshot of the MetricView tool for visualizing metrics data.

UML metrics

Software metrics are a well-established technique for managing source code quality during implementation.⁹ Metrics can also help manage the quality of architecture and design.

Figure 5 shows the extent of use that survey respondents indicated for metrics in each development phase, along with the extent to which they thought metrics would be useful. The desired use of metrics is two to four times higher than the actual use. These results emphasize the demand for measurement in UML models.

The multiple views of UML models include information not available from source code. This establishes a basis for architecture-level metrics.¹⁰ Projects could use these early metrics to indicate adherence to or violation of design guidelines and heuristics. The following describe some architecture metrics:

- **Class dynamicity.** This metric indicates a class's complexity. If a class has many different incoming and outgoing messages and appears in several different sequence diagrams, then we assume the class plays a critical role in the system and deserves more attention during reviews and testing.

If the dynamicity is above a threshold, the heuristic can recommend using a state machine to model the class's behavior.

- **Number of classes per use case.** This metric indicates whether related functionality is spread over many parts of a design. A higher value indicates low maintainability and reusability. Use cases with no classes indicate that the developers as yet foresee no implementation of the case, which might—in turn—indicate an implementation omission.
- **Number of use cases per class.** A class that is implemented in many use cases might have low cohesion and so serve a large amount of unrelated functionality. It might also be central to the system, which means that errors in it would cause many system features to suffer.

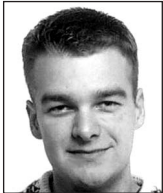
Existing tools usually present metrics and defects in tabular form. This requires designers to relate the tables to the diagrams they're working on. Presenting metrics values and defects graphically at the corresponding place in a UML diagram generates more direct feedback. In figure 6,

About the Authors



Christian F.J. Lange is a PhD student at the Eindhoven University of Technology and a researcher in the System Architecture and Networking group. His research interests include empirical software engineering and UML quality. He received his MSc in computer science from Eindhoven University of Technology. He's a member of the IEEE and the German Computer Society. Contact him at the System Architecture and Networking Group, Eindhoven Univ. of Technology, PO Box 513, 5600 MB Eindhoven, Netherlands; c.f.j.lange@tue.nl.

Michel R.V. Chaudron is an assistant professor at the Eindhoven University of Technology and a researcher in the System Architecture and Networking group. His research interests include software architecture, empirical software engineering, and component-based software engineering. He received his PhD in computer science from the Universiteit Leiden. Contact him at the System Architecture and Networking Group, Eindhoven Univ. of Technology, PO Box 513, 5600 MB Eindhoven, Netherlands; m.r.v.chaudron@tue.nl.




Johan Muskens is a researcher at Philips Research. His research interests include software architecture analysis, UML, component-based software engineering, and integration of third-party software. He obtained his MSc in computer science from the Eindhoven University of Technology. Contact him at Philips Research Laboratories, Office WDC 2.040, Prof. Holstlaan 4, 5656 AA Eindhoven, Netherlands; johan.muskens@philips.com.

our MetricView tool shows metrics and defects visualized on top of UML diagrams.

UML practices should improve with increased capabilities in development tools for it. Several areas need improvement:

- *Detection of design flaws, omissions, and inconsistencies.* Software metrics can play a role in detecting flaws, offering a fast, objective technique to support assessment of UML model properties. They also help identify refactoring opportunities to improve the architecture's structure.¹¹
- *More uniformity in modeling.* Modeling standards and development tools for checking them can achieve this purpose.
- *Domain- or project-specific reference architectures and patterns.* UML profiles support this work. UML 2.0 provides several facilities for defining such profiles but little support for checking them.
- *More consistency between UML models and system requirements as well as implementations.* Better mechanisms for traceability and round-trip engineering will help reduce these problems.

- *Defined quality goals for UML models.* A quality goal is an incentive to improve a model and identifies spots that must be improved to reach the defined goal. Existing UML development tools don't support the definition of measurable quality goals, and this functionality is needed.

The MetricView tool we applied to the case studies reported here makes analysis cost-effective by automating it. In most cases, the architects we talked with saw value in the analysis findings and acknowledged the tool's identification of weak spots in the models. In many cases, our feedback led to model changes to remedy identified problems. For some case studies, we've analyzed consecutive model versions and found improved model quality after rework. 

References

1. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.
2. C.F.J. Lange and M.R.V. Chaudron, "UML Practitioner Survey within the EmpAnAda Project," survey questionnaire; www.win.tue.nl/~clange/survey/survey.htm.
3. P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, vol. 12, no. 6, 1995, pp. 45–50.
4. C. Ghezzi and B. Nuseibeh, "Introduction to the Special Section: Managing Inconsistency in Software Development," *IEEE Trans. Software Eng.*, vol. 25, no. 6, 1999, pp. 782–783.
5. H. Sutter and A. Alexandrescu, *C++ Coding Standards*, Addison-Wesley, 2004.
6. M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed., Addison-Wesley, 2003.
7. C.F.J. Lange and M.R.V. Chaudron, "Effects of Defects in UML Models—An Experimental Investigation," to be published in Proc. IEEE/ACM Int'l Conf. Software Engineering (ICSE06), IEEE CS Press, May 2006.
8. P. Selonen and J. Xu, "Validating UML Models Against Architectural Profiles," *Proc. 9th European Software Eng. Conf. (ESEC)*, ACM Press, 2003, pp. 58–67.
9. N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, Int'l Thomson Computer Press, 1996.
10. J. Muskens, M.R.V. Chaudron, and C. Lange, "Investigations in Applying Metrics to Multi-View Architecture Models," *Proc. Euromicro*, IEEE CS Press, 2004, pp. 372–379.
11. M. Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.