

# Supporting task-oriented modeling using interactive UML views

Christian F.J. Lange\*, Martijn A.M. Wijns, Michel R.V. Chaudron

*Department of Mathematics and Computer Science, Eindhoven University of Technology,  
5600 MB Eindhoven, The Netherlands*

---

## Abstract

The UML is a collection of 13 diagram notations to describe different views of a software system. The existing diagram types display model elements and their relations. Software engineering is becoming more and more model-centric, such that software engineers start using UML models for more tasks than just describing the system. Tasks such as analysis or prediction of system properties require additional information such as metrics of the UML model or from external sources, e.g. a version control system. In this paper we identify tasks of model-centric software engineering and information that is required to fulfill these tasks. We propose views to visualize the information to support fulfilling the tasks. This paper reports on a large-scale controlled experiment to validate the usefulness of the proposed views that are implemented in our MetricView Evolution tool. The results of the experiment with 100 participants are statistically significant and show that the correctness of comprehension is improved by 4.5% and that the time needed is reduced by 20%.

© 2007 Elsevier Ltd. All rights reserved.

*Keywords:* UML; Interactive views; Quality; Task-orientation; Metrics; Comprehension; Empirical validation

---

## 1. Introduction

There is an ongoing trend of models becoming central artifacts in software development and maintenance. Originally models were mainly used during analysis and design of a software system. Nowadays the range of activities where models are used is much broader.

---

\*Corresponding author.

*E-mail addresses:* [C.F.J.Lange@tue.nl](mailto:C.F.J.Lange@tue.nl) (C.F.J. Lange), [MartijnWijns@gmail.com](mailto:MartijnWijns@gmail.com) (M.A.M. Wijns), [M.R.V.Chaudron@tue.nl](mailto:M.R.V.Chaudron@tue.nl) (M.R.V. Chaudron).

For example, the model driven architecture initiative (MDA [1]) has the goal to automatically generate the implementation from models. Other activities include but are not limited to effort estimation, prediction of quality properties, test case generation, and documentation. The de facto standard for modeling software is the unified modeling language (UML [2]) which is a graphical, multi-view notation. Therefore, we focus on UML in this study.

The building blocks of UML models are model elements, which are specified in the UML meta model [2]. Classes, methods, objects and messages are examples of UML model elements. These elements represent entities of software programs or relations between them. The UML has 13 diagram types to visualize UML models. Each diagram type describes a projection of a UML model from a certain perspective. Besides the model itself, today considerable amounts of related data are often available such as metrics [3,4], evolution data, documentation and problem reports.

The UML was originally intended to support designing software. However, as today's software development is becoming more and more model-centric, the UML experiences a much broader use. It is also used to understand systems during maintenance, as a starting point for testing activities, for predictions and other uses. We argue that in model-centric software engineering, views on the available data must be aligned with the tasks in which the views are used (similar to [5]). Most diagram types of the UML were adopted from other software modeling languages and are not aligned with all common tasks in model-centric software development. In this paper we use a framework which is the basis for developing views that support task-oriented modeling. We will identify typical tasks, available (and necessary) data, and we will propose new views and visualization techniques, to improve the use of UML models for practitioners with respect to fulfilling their tasks. We validated the task-oriented views using a large-scale controlled experiment. The purpose of the experiment was to find out whether the views improve various types of model comprehension tasks. The experiment consists of two runs with 100 participants each.

In Section 2 previous work that is related to this study is presented. In Section 3 the underlying concepts of the task-oriented modeling framework are described and the task-oriented views are presented. We report the validation in a large-scale controlled experiment in Section 4. Section 5 reports conclusions and presents directions for future work.

## 2. Related work

An approach related to ours is proposed by Kersten and Murphy [6]. Their tool MYLAR presents parts of a source code base which are relevant for a given task. The tool highlights program elements which have a high 'degree-of-interest' (DOI) and filters program elements with a low DOI. Their tool is adaptive in the sense, that the DOI-data are collected by the tool during usage. Hence, the interface adapts to a task, which is a difference to our approach, where we provide the user with pre-defined views for specific tasks. So far, their tool only takes internal properties into account. The main difference is, that their tool works for source code and we focus on UML models. Their tool's views are, in fact, adaptive interfaces, which change according to the usage pattern.

The SDMetrics tool [7] calculates metrics for UML models and presents the data in tables and graphs. The user still has to relate the metrics data to the model elements which

are on his mental map. As this mental map often coincides with the layout in the class diagrams, we propose the MetricView to combine metrics representation and the already existing layout of the class diagram. Similar visualizations to represent metrics are proposed by Langelier and Sahraoui [8]. Their visualizations mainly aim at visualizing metrics of source code, therefore they have to create a new layout, whereas we can use the already familiar layout of the UML class diagrams. In [9] several mappings from properties to visual properties, called polymetric views, are explored. The main difference to our work is that polymetric views are general software visualizations aiming at reverse-engineering tasks, while our work consists of UML-based visualizations targeted at various model-centric software engineering tasks. As most related work visualizes additional information such as metrics for source code, the work of Schauer et al. [10] also uses class diagrams as basis. They use color to indicate groups of classes belonging to the same design pattern [11]. Hansen [12] gives an application of the MetricView visualization: he visualized project data mapped to UML classes and packages and reports positive feedback from a case study within the company ABB.

Studies on layout algorithms are another direction of improving the visual representation of UML models. For example, the work of Eiglsperger [13] and Eichelberger [14] aims at creating layouts for class diagrams, which comply with rules from cognitive psychology. These layout algorithms are, in particular, useful for reverse engineered models, where no layout exists. An integral idea of MetricView is to keep the existing layout as created by the designer or generated by a layout algorithm. However, the idea of the Context View is to create a new class diagram only containing a class and its context. Kollmann and Gogolla [15] present a similar idea. They use metrics such as coupling, fan-in and fan-out to determine the set of classes which should be contained in their version of the Context View. In contrast to using metrics, we use the existing structure of relations in the model to determine the Context View.

In our literature study we found several studies about visualizing the evolution of source code, such as Voinea et al. [16] and Langelier and Sahrui [17]. However, only little work addresses the evolution of UML models. Xing and Stroulia [18] present the UMLDiff algorithm for automatic detection of structural changes between two class diagrams. The output of the UMLDiff algorithm is a detailed report of structural differences between two versions of UML class diagrams. Our Evolution View presents the trend of metrics of the entire model over several versions. Different from UMLDiff our Evolution View also takes external properties into account. However, opposed to UMLDiff, which is fully automated, our Evolution View is an aid for humans to analyze models.

The main purpose of our MetaView is to give an overview of and navigate through an entire UML model by showing all diagrams from all views concurrently. All UML case tools known to us only provide a tree for navigating through the model. Besides this, our literature study did not yield any work related to navigational aids for UML models.

The purpose of our experimental validation is to validate whether the proposed views support software comprehension tasks. To the best of our knowledge there exist no comprehension tools for software models. Therefore, we broaden the scope of the discussion of related work to comprehension tools for source code. Rigi [19] is a tool that visualizes the hierarchical structure of subsystems. Another tool is SHriMP [20] which visualizes the structure of a program in one window and provides techniques such as pan + zoom and fisheye-view to enable browsing detailed information as well as high-level

information. Our *Meta View* is a different visualization as it visualizes several UML diagrams and their relationships in one window, but it also enables switching between high and low level views using pan + zoom. Therefore, SHriMP and our Meta View support switching between top-down and bottom-up comprehension strategies [21]. The commercial software development SNiFF+ [22] rather provides functionality for browsing, cross referencing and searching source code, rather than visualizations. Storey et al. [23] compared SHriMP, Rigi and SNiFF in a controlled experiment. However, their experiment differs from the experiment described in this study: Storey et al.’s goal was to investigate how comprehension tools affect program comprehension. Storey et al. observed that many proposed tools lack an empirical validation of their effectiveness. Therefore, the goal of this study is to empirically validate the effectiveness of views of our MetricViewEvolution tool.

3. Task-oriented modeling views

3.1. A framework for task-oriented views

We aim at developing views that support tasks in model-centric software engineering. Therefore, we use a framework that relates tasks to views as a starting point to define the views. Maletic [24] proposed a framework to classify software visualizations focussing on the relation between tasks and views. In this study we reuse an adapted version of Maletic’s framework. In this section the three underlying concepts of the framework for task-oriented modeling and their relations are described. The three underlying concepts are: *Properties*, *Views* and *Tasks*. The relations between these concepts are illustrated in Fig. 1. Note some adjustments that we have made to Maletic’s framework: we refer to ‘view’ to describe the visualization and its representation, where Maletic’s framework has a separate dimension called ‘representation’. Additionally, we refer to ‘properties’ of a model, whereas Maletic uses the term ‘target’ for the same concept. For simplicity’s sake we omit in the framework the concept ‘audience’ which describes the stakeholder associated with a particular visualization and the dimension ‘medium’ which would be ‘color monitor’ in all cases.

Properties are characteristics of UML model elements (e.g. classes, use cases, or sequence diagrams). We will describe an initial overview of properties in this section. Then we define the concept of *views* which are the basis for visualizing the identified properties. A software developer has to fulfill *tasks* that have to be performed on model *properties* and that are supported by *views*. In the remainder of this section we explain the concepts in

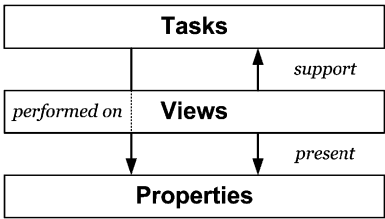


Fig. 1. The three underlying concepts and their relations.

more detail and provide examples for each concept. However, we do not claim that the given examples are exhaustive.

### 3.1.1. Tasks

We define a task in the context of this work as *a unit of work accomplished by a software engineer on a software artifact to fulfill a purpose*.

As software development is becoming more and more model-centric, the artifact of our interest is the (UML) model. Our proposed framework is a basis for developing model-centric views that support fulfilling the tasks. Hence, the tasks are the starting point for developing views. Note that our definition of a task does not define a task's granularity. In this section we give examples of task categories, that combine tasks with a common goal:

*Program understanding:* The need for program understanding (or model understanding) can have different reasons. One reason is that a new developer joins an existing team, and needs to understand the system before he is able to make useful contributions. Examples of activities related to this task category are: identifying key classes, identifying which classes implement which functionality, identifying relations between classes and identifying complex interactions.

*Model development:* Creation of models is often an incremental and iterative process including many changes. Either parts of the system are modeled in each step or the system as a whole is modeled from a high abstraction level down to a more detailed level. Examples of activities employed in this type of task are: adding, changing or removing elements.

*Testing:* Testing is used to detect defects in software. A testing task common in model-centric software development is the (automatic) generation of test cases from sequence diagrams.

*Model maintenance:* The process of changing a software system due to corrections, improvements or adjustments is called maintenance. The tasks related to changes in the model belong to this task category. Some activities in which UML models are involved include: extension of a system, bug fixing, handling change requests and performing impact analysis before making a change.

*Quality evaluation:* As the correction of quality problems is much cheaper at an early stage, i.e. at the modeling stage, than at implementation stage, it is important to evaluate a system's quality before implementing it. The evaluation of the quality of a model can be performed at several abstraction levels, e.g. separate elements, diagrams or the system as a whole. Besides evaluating a single version of a model one can also investigate multiple versions at once, to detect trends.

*Completeness/maturity evaluation:* Related to quality evaluation is the evaluation of the completeness or the maturity of a model. This task category includes analyzing whether a model reflects all requirements and whether the diagrams of the model describe the system completely.

### 3.1.2. Properties

We define a property in the context of this work as *a directly or indirectly measurable characteristic of a model element*. The set of its properties (or a subset of it) uniquely identifies a model element. Properties belong to the information needed by software engineers to perform tasks. Model elements are the building blocks of UML models. The model element types are defined in the UML meta model. For each model element type,

such as class, association, classifier instantiation, use case, etc., a number of properties are defined. We identify three different types of properties for model elements:

*Direct internal:* Those properties of an element that are solely and directly based on information that is present in the model. General examples of this kind of property are the name of an element or the owner of an element. Example properties for a class are its operations, its attributes and its relations to other classes.

*Indirect internal:* Besides the information that is directly present within the model, we identify properties associated with model elements that can be derived from the model. The information contained in indirect internal properties is derived from the model element itself or related model elements that may also be in different diagrams. For example, multi-view metrics as proposed in [25] such as ‘number of use cases per class’ are indirect internal properties that combine information from different diagrams. General examples of this kind of property are metrics and history data. Other examples of indirect internal properties of the model element ‘class’ are the number of methods, the number of instantiations of the class or the complexity of the class (for example, based on an associated state diagram).

*External:* A third type of property is based on information from outside the model. This type of properties is ignored by the UML specification and to the best of our knowledge there exist no commercial tools that take external properties of model elements into account. Sources of external properties are other artifacts, such as source code, requirement documents or test documents. Additionally, data that are recorded during software development which is traceable to model elements are an external property. For example, bug reports can be seen as external properties of classes or can even be traced to use cases. Configuration management systems allow to collect evolution data about a software artifact. For example, the number of changes or the number of different developers working on a model element are external properties. As UML is used nowadays not only in design, but also in other phases, such as maintenance, external property data are available for UML models.

### 3.1.3. Views

We define a view in the context of this work as *a visible projection of a subset of properties of a model’s elements to fulfill a task*. Typically, UML models are visually represented in diagrams. The UML specification [2] defines a variety of diagram types as views on a model from a certain perspective. This specification is only concerned with internal properties and not even all of these properties are viewable in UML diagrams. The relations between model elements in different diagram types are often not intuitively presented by UML. It is, for example, often difficult and tedious to find out on which places a class is instantiated in the model, because this relation is not explicitly present in the diagrams. We argue that in the design of the UML the choice of which properties can be viewed in UML diagrams and the visualization techniques used to represent them are not optimal for common tasks in software engineering. Views offer visual representations of a model. In a view model element properties are visualized by creating a mapping to visual properties. Examples of these visual properties are: position (layout), size (width, height, depth), color (hue, saturation, luminance), shape and orientation.

### 3.2. MetricViewEvolution

This section describes the interactive views we proposed to support comprehension of UML models. In addition to the textual description, the views are summarized according

to the task-oriented modeling framework in Table 1. The views are implemented in our tool *MetricViewEvolution*.

### 3.2.1. MetaView

Fig. 2 shows a MetaView in which inter-diagram relations are visualized. The different UML diagrams that are created during the software engineering process offer different

Table 1  
Description of the views according to the task-oriented modeling framework

View	Task	Properties
MetaView	Program understanding, maintenance, completeness analysis	Implicit and explicit relations between diagrams
MetricView	Program understanding, quality evaluation, completeness evaluation	Metrics (direct and indirect internal, external)
UML-City View	Program understanding, quality evaluation, completeness evaluation	Metrics (direct and indirect internal, external)
ContextDiagram	Program understanding	Implicit and explicit relations between diagrams
QualityView	Quality evaluation	Metrics (direct and indirect internal, external)
EvolutionView	Quality evaluation, identification of trends	Metrics (direct and indirect internal, external), different versions

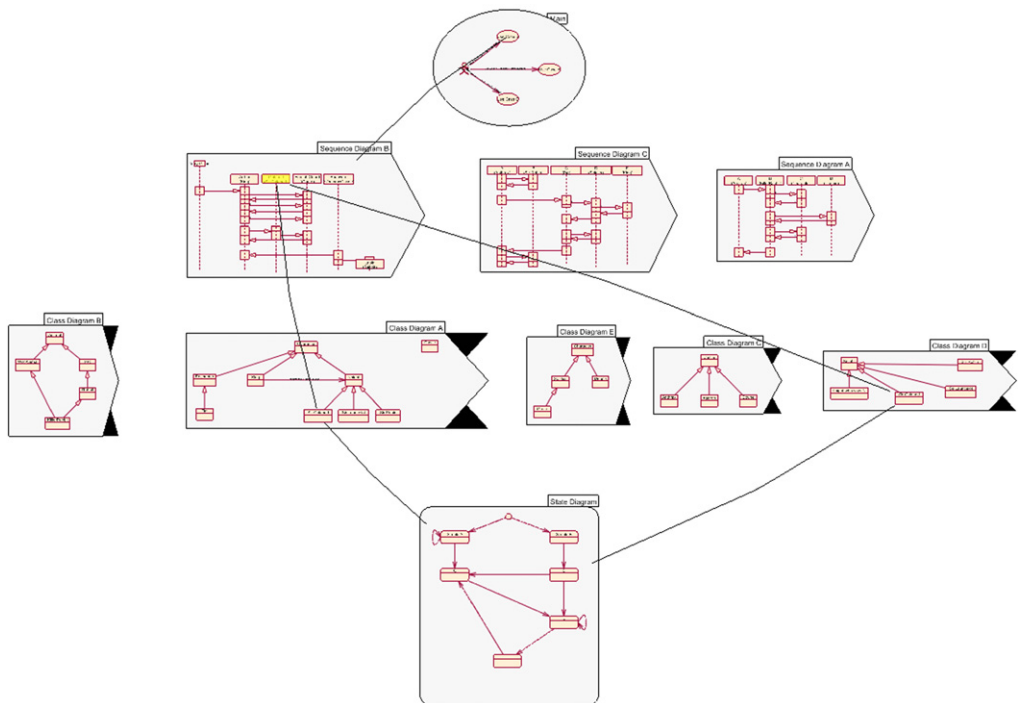


Fig. 2. MetaView: tracing from an object to a use case and to a related class and state diagram.

levels of abstraction. Typically, starting at a high abstraction level we find the use cases. These use cases are described in more detail by sequence diagrams. In sequence diagrams occur objects representing classes. Classes and their relations to each other are described in class diagrams. The internal behavior of these classes can be described by state machines.

A problem that exists in regular UML tools is that each of the diagrams is shown separately; this results in the hiding of the relation between different diagrams and model elements. Our proposed solution to these problems is the MetaView. It gives an overview of the diagrams that describe the model and makes it possible to show the relations between (elements on) different diagrams. This last feature allows tracing through the different abstraction levels that the different types of diagrams offer.

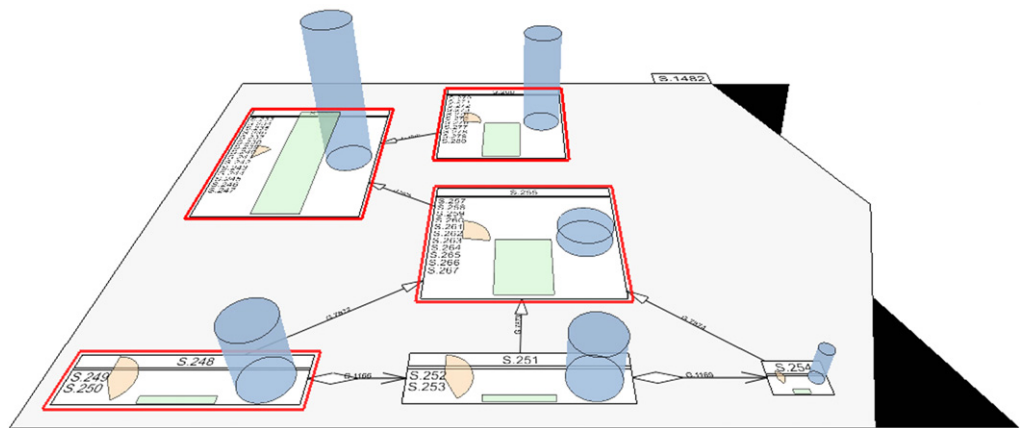
Fig. 2 shows the four types of elements that take part in this example: a use case, an object that occurs in the sequence diagram describing the use case, the object’s class, and the state diagram describing the class.

3.2.2. *MetricView*

Fig. 3 shows an example of the proposed MetricView, in which three different metrics are visualized on top of a regular class diagram. The idea of MetricView is to combine the existing layout of UML class diagrams with the visualization of metrics and UML models using a set of techniques adopted from geographical information systems (GIS) [26]. Applying metrics to a UML model can result in an overwhelming amount of data. This data are usually presented in tables, such that the software engineer has to make the mapping between metrics values in the table and classes in the UML diagrams manually or in his mind. MetricView overcomes this problem by integrating the model and metric visualization. MetricView can represent the metrics using color, size and/or shape to visualize values.

3.2.3. *UML-City View*

Fig. 4 shows an example UML-City. This view combines the concepts of the MetaView and MetricView. As metric visualization the ‘3D-heightbar’ is used, this visualization shows a box on top of the model element where the height and the color of the box indicate the value of the metric. Low metric values are depicted by flat green boxes while high



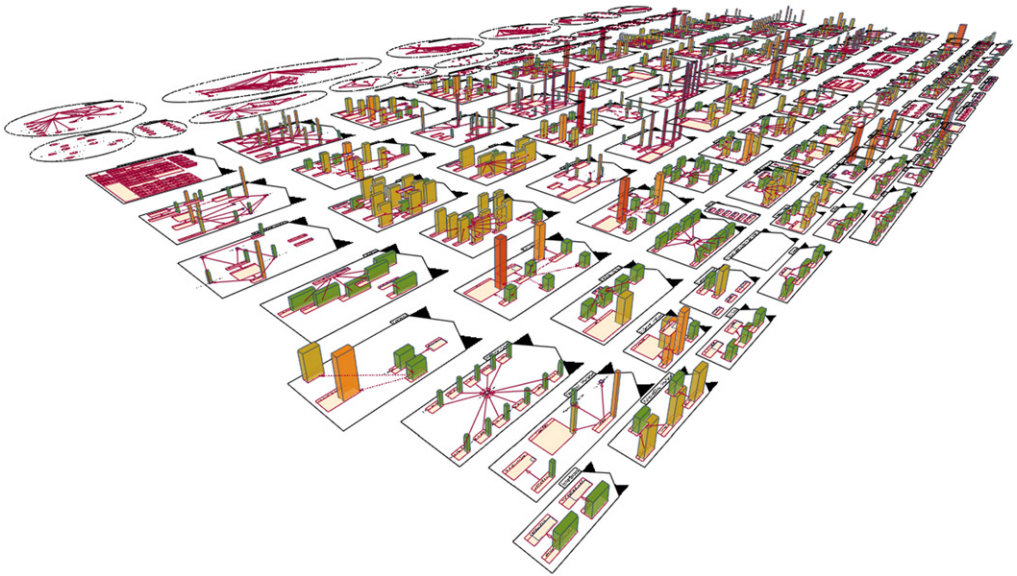


Fig. 4. UML-City: combining the MetaView and MetricView.

values are depicted by tall red boxes. Applying this to the MetaView results in a view for which city is a metaphor.

#### 3.2.4. Context View

The context of a model element consists of all model elements it relates to. The elements of a model are typically scattered over several diagrams. UML diagrams are projections of the entire model, they typically do not contain all model elements. Accordingly, it often occurs that only a limited context of model elements is viewed in one diagram. To fully understand a model element it might be necessary to know its entire context. Therefore, we propose the *Context View*. The model element whose context is viewed in a context view is centered in the diagram. All model elements that are directly related to the particular model element are viewed as a circle around the model element. As the context of a model element is potentially very large and only a specific subset of the context might be necessary for a task it is desirable to filter the context. Two straightforward filtering criteria are the model element type or the relation type. Fig. 5 shows the context view for a class. The context is filtered such that only model elements of the type ‘class’ that are related to the class via an ‘inherits from’ relationship are viewed. The right side of the of Fig. 5 contains one of the class diagrams as shown in current UML tools. Only four classes that inherit from the particular class are shown.

#### 3.2.5. Quality Tree View

Quality models such as ISO 9126 [27] and proposed by Khosravi et al. [28] provide a structure to define what quality means in a given context. The most common approach to create such models is used in so-called decompositional quality models. Quality is decomposed in subconcepts such as maintainability and understandability. Each of these

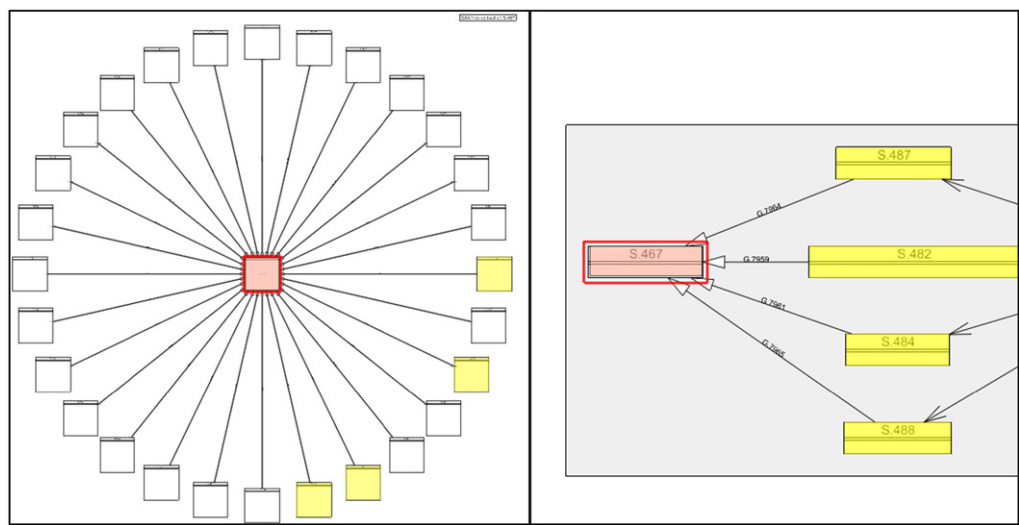


Fig. 5. Context View (left) showing all the children of a single class, compared with regular class diagram (right).

subconcepts can in turn be decomposed again. In this way a tree-like structure is constructed which builds a quality model. At the leafs of the tree are metrics. These metrics may be obtained from a UML model and the metric values are used to calculate values for each of the higher nodes in the quality model.

In the Quality Tree view, the nodes represent the concepts of the quality model and the edges represent the relations between the concepts. Our contribution to the well-known concept of quality models is that in the Quality Tree view, color or graphs represent the value of each node. Additionally, our proposed Quality Tree View is implemented, such that it interacts with the UML model and supports traceability between nodes in the quality model and elements in the UML model.

As quality models will differ based on the context in which they are used, the Quality Tree View is configurable to represent any hierarchical quality models, such as the quality model for UML [29]. This tailoring is possible by changing the structure of the tree, as well as by changing the functions attached to the relations in the tree. Additionally, it is possible to change the metrics that produce the input for the quality model.

3.3. Evolution View

Fig. 6 shows the *Evolution View*, in which the two concepts graph and calendar are combined to identify trends. The reason for using a graph is that it is an effective way to visualize the evolution of metric data. The purpose of the *Evolution View* is to enable users to spot trends in the values of quality attributes and/or metrics at multiple abstraction levels. At system level such a graph can be used to give an overview of changes in aggregated data. By combining it with the concept of a calendar, i.e. mapping time on the horizontal axis and values of the vertical axis, and adding color to indicate whether a given value is considered good or bad it becomes a compact and intuitive way to enable the evaluation of the evolution of quality data. The same

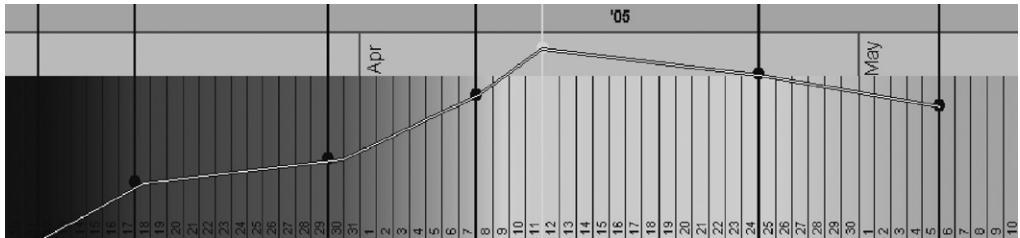


Fig. 6. Evolution View: based on the calendar metaphor.

technique can be applied at diagram and element level to allow for different analysis granularity.

### 3.3.1. Search and highlight

The aforementioned views are complemented with search and highlight functionality. The results of search actions are visualized by highlighting the relevant diagrams and diagram elements with a distinct color (e.g. yellow). Additionally, relations between elements in the set of result and other elements in the model are drawn.

## 4. Validation using a controlled experiment

### 4.1. Design of the experiment

#### 4.1.1. Purpose and hypotheses

We have proposed new views for analyzing UML models as described in Section 3. The purpose of this experiment is to validate the proposed views. As the available time and resources are limited, we limit the experimental validation to the following views: MetaView, ContextView, MetricView, UML-CityView, and the search and highlight functionality.

We summarize the purpose of the experiment according to the goal-question-metric paradigm (GQM) [30] as follows:

*Analyze Comprehension of UML views  
for the purpose of evaluation  
with respect to correctness and effort  
from the point of view of the researcher  
in the context of Master's students at the TU Eindhoven.*

More specifically, we want to compare the usefulness of the proposed views with the traditional views, which are used as a baseline in this experiment. We limit the scope of this experiment to comprehension tasks. Validation of other tasks should be conducted in future work. Comprehension plays an important role in the development and maintenance of software systems. In particular, two concepts are of interest for the evaluation of understanding techniques: correctness and effort. Correctness is essential for understanding. Incorrect understanding of a system can lead to wrong actions introducing faults and communication overhead. Effort is relevant from an economical point of view.

To evaluate the proposed view we are interested whether they differ from the traditional UML views. This leads us to the following hypotheses (the alternative hypotheses are the negations of these null-hypotheses):

- H1<sub>0</sub>: There is no significant difference between proposed and traditional views in terms of correctness of understanding the model.
- H2<sub>0</sub>: There is no significant difference between proposed and traditional views in terms of effort needed for understanding the model.

#### 4.1.2. Task, objects and treatment

The task in the experiment was to answer comprehension questions about a UML model. In order to answer the questionnaire, the subjects had to analyze a given UML model using a tool. The experiment was carried out in two runs. In each of the two experimental runs, a different UML model had to be analyzed. The questionnaires for both runs were conceptually equal and contained 29 multiple-choice questions of the following categories:

*Category 1: Multiple class diagrams (CDs).* The questions address information that is scattered over multiple class diagrams. Subcategories address coupling via associations, inheritance and occurrences of classes in several class diagrams. This category contains six questions.

*Category 2: Relations between class diagram and sequence diagram (CD–SD).* The questions address relations between elements in class diagrams and sequence diagrams. Subcategories address message calls between class instantiations and occurrence of classes in sequence diagrams. This category contains nine questions.

*Category 3: Relations between use cases, sequence diagrams and class diagrams (UC–SD–CD).* The questions address relations between elements in use cases, sequence diagrams and class diagrams. Examples are, which classes contribute to a particular use case. This category contains six questions.

*Category 4: Metrics.* The questions address metrics of the UML model. This category contains eight questions, of which four purely address metrics and four address metrics combined with a structural relation between elements.

Examples of questions are: ‘Which classes contribute to implementing the use case “Leave Route”?’ (Category 3) and ‘Which classes receive method calls of class “Route”?’ (Category 2). The entire questionnaires are available in the replication package [31].

The objects in this experiment are the UML models that are used for the analysis task. Both models were of similar size and complexity: 39 resp. 38 classes, 11 use cases, 5 class diagrams, and 5 resp. 6 sequence diagrams. The size was chosen to be larger than a pure toy-example, but still allowing to fulfill the task in the given amount of time. The application domains were an insurance information system in the first run and a car navigation system in the second run. It is reasonable to assume that both application domains were equally familiar to the subjects.

The treatment in this experiment is the use of the tool MetricView Evolution which implements the views discussed in Section 3. The control group uses a combination of case tools that represent the current state-of-the-practice in UML tools: Poseidon (<http://www.gentleware.com>) and SDMetrics [7]. Poseidon is a user-friendly UML case tool which is used to create and view UML models. Its features are similar to most popular

UML case tools including Rational Rose. SDMetrics is an established metrics analysis tool for UML models.

#### 4.1.3. Subjects and design

In total 100 MSc students participated in the experiment, which was conducted within the course “Software Architecting” in the fall term of 2006 at the Technische Universiteit Eindhoven (TU/e). The 80% of the students have a bachelor degree in computer science, the other subjects have a bachelor degree in related disciplines. The students have experience using the UML and related tooling. The students were not familiar with the goal and the underlying research question of the experiment to avoid biased behavior. The subjects’ prior education was received from universities within and outside Europe.

The design of the experiment is a between-subjects design as described in Table 2. The 100 subjects are assigned to two groups (A and B) by randomization, such that we can assume absence of variation [3] in subject background between the two groups. To avoid learning effects of the tool or the UML model, the groups had to switch tools and a different UML model was used as object in the second run. Note that for the second run five subjects of the first run did not show up.

#### 4.1.4. Preparation and operation

Before the experiment we conducted a pilot run to evaluate and improve the experimental material. The participants of the pilot run did not participate in the main experiment.

Both runs of the experiment were carried out as an assignment in an exam-like setting. The subjects performed the task individually. The subjects used their own laptops with the tools installed and ready to run. We distributed the models using our university’s education-support system ‘StudyWeb’ (<http://studyweb.tue.nl/>). To prevent subjects from switching between treatments, i.e. tools, we configured the StudyWeb system such that for each subject the model was only available in the file format for the tool it was supposed to use. The questionnaires were distributed as hardcopies, as for both treatment groups the questionnaire was the same. Based on the experience from the pilot run the duration for the experiment was set to 120 min. The subjects were not allowed to communicate verbally or via network communication. This was enforced by spot checks.

#### 4.1.5. Variables

The independent variables in this experiment are tool ( $L$ : MetricViewEvolution (MVE) or Poseidon + SDMetrics (PoS)), run ( $R$ : 1 or 2) and UML model ( $M$ : insurance information system (IIS) or car navigation system (CNS)). As described above the variables  $R$  and  $M$  coincide in our experimental design.

Table 2  
Experimental design

	MVE	POS
First run	Group B (48)	Group A (52)
Second run	Group A (50)	Group B (45)

As described above we are interested in the correctness of and the effort needed for the comprehension task. This leads to the following two dependent variables:

- Time  $T$ : total time in minutes to perform the task.
- Correctness  $C$ :  $\frac{\text{number of correct answers}}{\text{total number of questions}}$ .

We chose to measure correctness as a ratio instead of the absolute number of correct answers to make the results easily comparable to replications of this experiment with a different number of questions.

Now we can express the hypotheses in terms of the variables (where in  $T_x$  and  $C_x$  are the time and correctness of the task performed with tool  $x$ ):

- $H1_0 : T_{MVE} = T_{POS}$ .
- $H2_0 : C_{MVE} = C_{POS}$ .

In the experiment we used the paper questionnaires for data collection. The time was logged by the students in slots on the questionnaire. The subjects were instructed to complete the questions in the given order, to log breaks, and to start when they started answering the questions after the tools had been set up.

#### 4.1.6. Analysis techniques

We summarize the results of the experiment using the established descriptive statistics. For hypothesis testing we use the Mann–Whitney test, which is suitable for a between-subjects design. We used the tool SPSS (<http://www.spss.com>). To enable comparison of our results with other studies we report the effect size using Cohen's  $d$  [32]. The subjective evaluation data are obtained from the post-test questionnaire. The answers are on a five-point Likert-scale and, hence, measured on an ordinal scale. We summarize the data by presenting the frequencies as percentages for each answer option and providing additional descriptive statistics where appropriate. We compare the equality of answer distributions between different treatment groups using the  $\chi^2$ -test [33]. For this test we used Microsoft Excel. We apply the standard threshold of  $p < 0.05$  for statistical significance. When we compare distributions a  $\chi^2$  value less than 7.82 implies that  $p < 0.05$ .

## 4.2. Results

Here we present the results including a discussion of outlier analysis, descriptive statistics, hypothesis testing and participants' subjective evaluation.

### 4.2.1. Outlier analysis

Outliers are extreme values that may influence the conclusions drawn from the collected data. To be able to draw valid conclusions from the data Wohlin et al. [34] suggest to remove outliers caused by exceptional conditions that are unlikely to happen again from the data. Outliers without an exceptional cause should stay in the data. Using boxplot analysis we identified 10 outliers. Further analysis led to the exclusion of four outliers from the data set, because they were caused by exceptional conditions such as technical problems.

Table 3  
Descriptive statistics and results of hypothesis tests

Dep. Var.	Tool	$N$	Mean	%	$d$	$p$ (M.–Whitn.)	Hypothesis
First run							
Time	MVE	45	54.62	80.81	0.90	<0.001	H1 <sub>0</sub> rejected
Time	POS	51	67.59	100.00			
Corr.	MVE	45	0.838	104.77	0.45	0.041	H2 <sub>0</sub> rejected
Corr.	POS	51	0.799	100.00			
Second run							
Time	MVE	48	52.48	79.41	1.09	<0.001	H1 <sub>0</sub> rejected
Time	POS	43	66.09	100.00			
Corr.	MVE	48	0.921	104.54	0.70	0.002	H2 <sub>0</sub> rejected
Corr.	POS	43	0.881	100.00			

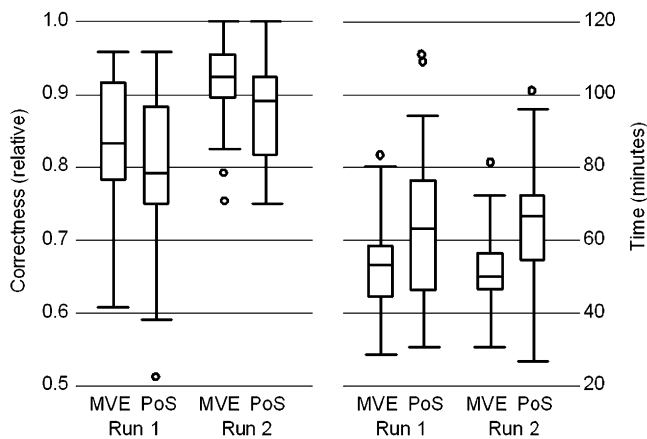


Fig. 7. Boxplots for correctness and time.

#### 4.2.2. Descriptive statistics

The data of the dependent variables *correctness* and *time* are summarized in Table 3. The data are presented separately for the first and the second run. The columns of the table show the treatment, the number of correct observations (*N*), the mean, the relative difference of the mean between the MVE group and the control group (percentage), and for hypothesis testing the *p*-value of the Mann–Whitney test and whether the corresponding hypothesis was rejected or not. Additionally the data are summarized in boxplots in Fig. 7.

#### 4.2.3. Testing hypothesis H1—Time

Table 3 shows that the Mann–Whitney test yields *p*-values below the significance threshold of 0.05. Therefore, we can reject the hypothesis H1<sub>0</sub> which states that there is no difference in time for completing the task using MVE or PoS. The effect size is 0.90 in the first run and 1.09 in the second run. This is a relatively large effect since Cohen [32] states that an effect size of 0.8 is ‘large’. The descriptive statistics in Table 3 show difference in

time for both runs. On average the MVE users use around 20% less time for analyzing the UML model than the control group.

One could argue that the time difference in favor of MVE is caused by the fact that the PoS group used two different tools and switching between tools is time consuming. However, a detailed look at the results shows that this is not the case. Switching between tools would only be required to answer the questions of one of the four question categories (only the questions regarding metrics). Therefore, the time difference should mainly occur for the questions of the category ‘metrics’. But the opposite is the case: the effect size, measured in Cohen’s *d*, is small for the metrics questions (*d* is approximately 0.3) and larger for the questions of the other categories (*d* is approximately 1.0). Hence, we conclude that the results for time are not biased due to the fact that the treatment PoS comprises the use of two separate tools.

To find out whether the time difference is larger for category 4 than for the other categories we used the Mann–Whitney test and we measured the effect size.

4.2.4. Testing hypothesis H2—Correctness

The results for testing the hypothesis concerning correctness of the comprehension task (H2) are also presented in Table 3. For both experimental runs there is a significant difference in correctness of the task between MVE users and the control group. As the statistical tests yielded *p*-values far below the significance level of 0.05, we reject H2<sub>0</sub>. The effect size is approximately 0.5 in the first run and 0.7 in the second run, which is a medium effect according to Cohen [32]. Table 3 describes the differences between the mean values of the MVE group and the control group. In the first run the mean was increased with 4.77% by using MVE and in the second run the increase was 4.54%.

Table 4  
Subjective Evaluation (\* indicates significant difference)

ID	Name	Run 1			Run 2		
		MVE	PoS	$\chi^2$	MVE	PoS	$\chi^2$
Model							
M1	Quality	3.09	3.20	1.37	3.41	3.39	0.74
M2	Understandability	3.34	3.10	2.81	3.49	3.28	2.27
M3	Completeness	2.37	2.55	1.56	3.10	2.95	2.96
Task							
S1	Understandability	3.98	3.41	*13.54	4.00	3.77	4.14
S2	Difficulty	3.65	3.24	*19.18	3.71	3.48	3.66
S3	Enjoy	3.49	3.08	6.29	3.40	3.07	3.21
S4	Motivation	3.89	3.80	4.39	3.63	3.59	4.01
Tool					Preference in %		
T1	Model Understanding	3.77	3.49	6.52	80.9	19.1	
T2	Quality Eval.	3.68	3.31	7.42	78.4	21.6	
T3	Metrics Analysis	3.96	3.76	1.51	60.9	39.1	
T4	Navigate through Model	3.43	3.29	6.97	71.6	28.4	
T5	Usability	3.43	3.50	2.42	74.7	25.3	

#### 4.2.5. Subjective evaluation

The results of the subjective evaluation are summarized in Table 4. Each row corresponds to one question of the questionnaire's subjective evaluation section. The first two columns contain the questions internal identifier (ID) and a descriptive name. The questions were multiple choice questions on a five-point Likert scale with the following levels: 1 = 'very poor', 2 = 'poor', 3 = 'medium', 4 = 'good' and 5 = 'very good'. The table summarizes the answers for each question using the mean. To find out whether there are statistically significant differences between the answers of the treatment groups we report the  $\chi^2$ -test results. We indicate significant results in the table using an asterisk (\*). Note that in the second run the question type for questions about the tool was changed to binary preference questions, therefore we report the percentages.

*Evaluating the model.* The three questions about the model addressed model quality, model understandability and model completeness, respectively. The results are around medium and reflect the authors' evaluation of the UML models. There were no significant differences between the evaluations of the two treatment groups.

*Evaluating the task.* For motivation and for understandability the mean values range from 'medium' to 'good'. For motivation there are no significant differences. It is important to have motivated subjects who understand the task well in order to obtain valid results from the experiment. However, for understandability and for perceived difficulty we have a significant difference in the first run. As the task was the same for both treatment groups, we assume that the subjects include the understandability and difficulty of using the tool in this question. The understandability and difficulty results improved in the second run. The level to which the subjects enjoyed the task is between 'medium' and 'good'. In both runs the mean of the MVE group is higher than the control group's mean, but not statistically significant. A good level of enjoyment (combined with an acceptable level of perceived difficulty) is an important factor for successful adoption of a new technology in practice [35], as enjoyment in using the technology increases the intrinsic motivation to use it.

*Evaluating the tools.* In the first run the subjects rated the expected suitability of the tool they used for four activities and tool usability. The results in Table 4 show that both tools are rated between 'medium' and 'good' on average.

We complement this analysis with a direct comparison of both tools in the second run. The subjects had to indicate which tool they prefer. The vast majority of the subjects prefers MVE for the four activities.

### 4.3. Threats to validity

Conducting experiments involves threats to the validity of the results. In this section we discuss how we deal with the potential threats to the validity of the reported experiment. We structure our discussion according to Wohlin et al. [34] into internal validity, external validity, construct validity and conclusion validity.

#### 4.3.1. Internal validity

Threats to internal validity can affect the independent variables of an experiment. A possible threat to internal validity is that the treatment groups behave differently because of a confounding factor such as difference in skills, experience or motivation. We used randomization to assign subjects to treatment groups to avoid differences

between treatment groups. There were no significant differences in subjects' background and motivation (see Sections 4.1.3 and 4.2.5).

The experiment was conducted in a controlled environment, i.e. an exam-like setting. Due to limited room capacity the risk for cheating was somewhat higher in the second run, but due to the taken precautions and spot checks we regard the chance of biased results due to cheating as very small. Another possible threat to validity is that subjects used a different tool for the analysis and therefore did not adhere to the treatment. We eliminated this threat by providing the subjects with the model in the file format of the tool belonging to their treatment group. Additionally, we conducted spot checks during the experiment and found no subject deviating from its treatment.

#### 4.3.2. External validity

Threats to external validity reduce the generalizability of the results to industrial practice. The experiment is designed to render a realistic situation. We use students as subjects, which might be a threat to external validity. However, Kitchenham et al. [36] state that students can be used as subjects. All students in this experiment have relevant experience, hold a BSc degree and have received their BSc from different universities and different countries. The objects in this experiment are UML models that describe systems from realistic application domains. The consistent results of both runs support that the observed effect can be expected in models of different application domains. The models are larger than 'toy models' but in practice models of much larger size are used. However, we assume that support for comprehension is even more important for larger models. Therefore, we assume that the observed effect for larger models will be at least as large as it is observed in this experiment.

A possible issue is the representativeness of the experimental task. To the best of our knowledge there exist no published research results describing comprehension tasks for UML models. Pacione et al. [37] conducted literature survey to compile a list of general comprehension activities. The comprehension questions of our experiment cover different conceptual categories. These categories can be related to the activities in Pacione's list. Some of the activities listed by Pacione are not covered in this experiment. The uncovered activities are mainly related to information that is normally not present in practice in UML models, such as runtime information.

#### 4.3.3. Construct validity

Construct validity is the degree to which the variables measure the concepts they should measure. The dependent variables in this experiment are comprehension correctness and time. Comprehension correctness is difficult to measure as it is closely related to the representativeness of the comprehension task as discussed above. However, the measurement of correctness is objective and repeatable as it is based on a multiple-choice questionnaire which is provided in the replication package [38]. The time is measured by logging the actual time in predefined spots on the questionnaire as described in Section 4.1.5 and we do not expect the results for time being biased.

#### 4.3.4. Conclusion validity

Conclusion validity is concerned with the relation between the treatment and the outcome. The statistical analysis of the results is reliable, as we used robust statistical methods. The results are statistically significant with very low  $p$ -values.

We minimized possible understanding problems by testing the experiment material in a pilot experiment and improving it according to the observed issues. The course instructors were available to the students for clarification questions. The results of the post-test questionnaire show that the task was well understood. Hence, we conclude that there were no understanding problems threatening the validity of the reported experiment.

## 5. Conclusions

In this paper we state the problem that existing representations of UML models are not sufficient for common model-centric software engineering tasks. We use a framework consisting of UML model elements, their properties, and software engineering tasks, that form a basis to develop new views of UML models and related information. Based on this framework we propose eight views to support different tasks. The views are implemented in our MetricView Evolution tool. Instead of performing fully automated analysis of UML models, the presented tool supports the user by presenting information required for a task.

We have validated the proposed views a large-scale controlled experiment. In the experiment we compared the performance on comprehension tasks supported by our tool MetricViewEvolution (MVE) with the performance on comprehension tasks supported by a traditional UML CASE tool (Poseidon) and metrics tool (SDMetrics). The results show that the effort needed is reduced by approximately 20% and the correctness of the comprehension task is increased by approximately 4.5%. These results are statistically significant. Additional analysis reveals that the participants prefer our MVE tool above traditional tooling. This indicates that the adoption threshold for our MVE views is low.

Our proposed views are simple and straightforward. They should be improved by established techniques from information visualization such as mechanisms for filtering and abstraction and by using more sophisticated layout algorithms. As our simple implementation already led to considerable improvements we conclude that the field of improving the comprehension of UML models is a promising direction for future research. As software development and maintenance is becoming more model-centric, we should also shift our research efforts into this promising direction.

In future work the proposed views should be integrated into model editors instead of pure model viewers. Using the enhanced editors a similar validation should be conducted where the set of tasks also includes change tasks instead of pure comprehension tasks.

## References

- [1] Object Management Group, MDA Guide, Version 1.0.1, omg/03-06-01 ed., June 2003.
- [2] Object Management Group, Unified Modeling Language, UML 2.0 Superstructure Specification, formal/05-07-04 ed., July 2005.
- [3] N.E. Fenton, S.L. Pfleeger, *Software Metrics, A Rigorous and Practical Approach*, second ed., Thomson Computer Press, Boston, MA, 1996.

- [4] S.H. Kan, *Metrics and Models in Software Quality Engineering*, second ed., Addison-Wesley Professional, Reading, MA, 2002.
- [5] G.C. Murphy, M. Kersten, M.P. Robillard, D. Čubranić, The emergent structure of development tasks, in: A.P. Black (Ed.), *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP 2005)*, Lecture Notes in Computer Science, Springer, Berlin, 2005.
- [6] M. Kersten, G. Murphy, Mylar: a degree-of-interest model for IDEs, in: *Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD 2005)*, 2005.
- [7] J. Wüst, The software design metrics tool for the UML (<http://www.sdmetrics.com>).
- [8] G. Langelier, H. Sahraoui, P. Poulin, Visualization-based analysis of quality for large-scale software systems, in: *ASE'05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005.
- [9] M. Lanza, S. Ducasse, Polymetric views—a lightweight visual approach to reverse engineering, *IEEE Transactions on Software Engineering* 29 (9) (2003) 782–795.
- [10] R. Schauer, R.K. Keller, Pattern visualization for software comprehension, in: *Proceedings of the 6th International Workshop on Program Comprehension (IWPC'98)*, Ischia, Italy, 1998.
- [11] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1994.
- [12] K.T. Hansen, Project visualization for software, *IEEE Software* 23 (4) (2006) 84–92.
- [13] M. Eiglsperger, Automatic layout of UML class diagrams: a topology-shape-metrics approach, Ph.D. Thesis, Universität Tübingen, Germany, 2003.
- [14] H. Eichelberger, Aesthetics and automatic layout of UML class diagrams, Ph.D. Thesis, Fakultät für Mathematik und Informatik, Würzburg University, Germany, 2005.
- [15] R. Kollmann, M. Gogolla, Metric-based selective representation of UML diagrams, in: T. Gyimothy, F.B. e Abreu (Eds.), *Proceedings of the 6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, IEEE, Los Alamitos, 2002.
- [16] L. Voinea, A. Telea, J.J. van Wijk, CVSScan: visualization of code evolution, in: *Proceedings of the ACM Symposium on Software Visualization SoftVis*, ACM, St. Louis, Missouri, 2005.
- [17] G. Langelier, H.A. Sahraoui, Animation coherence in representing software evolution, in: *Proceedings of (QAOOSE'06)*, 2006.
- [18] Z. Xing, E. Stroulia, UMLDiff: an algorithm for object-oriented design differencing, in: D.F. Redmiles, T. Ellman, A. Zisman (Eds.), *Proceedings of 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, ACM, New York, 2005.
- [19] M.-A.D. Storey, K. Wong, H.A. Müller, Rigi: a visualization environment for reverse engineering, in: *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, IEEE Computer Society Press, Silver Spring, MD, 1997.
- [20] M.-A.D. Storey, C. Best, J. Michaud, SHriMP views: an interactive environment for exploring javaprograms, in: *Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*, IEEE Computer Society Press, Silver Spring, MD, 2001.
- [21] A. von Mayrhauser, A.M. Vans, Program comprehension during software maintenance and evolution, in: *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, IEEE Computer Society Press, Silver Spring, MD, 1997.
- [22] SNIFF+ 3.0, TakeFive Software GmbH, 1997.
- [23] M.-A.D. Storey, K. Wong, H.A. Müller, How do program understanding tools affect how programmers understand programs?, *Science of Computer Programming* 36 (2–3) (2000) 183–207.
- [24] J.I. Maletic, A. Marcus, M.L. Collard, A task oriented view of software visualization, in: *Proceedings of the IEEE Workshop of Visualizing Software for Understanding and Analysis (VISSOFT 2002)*, IEEE Computer Society, Silver Spring, MD, 2002.
- [25] C.F.J. Lange, M.R.V. Chaudron, J. Muskens, In practice: UML software architecture and design description, *IEEE Software* 23 (2) (2006) 40–46.
- [26] C.F.J. Lange, M.R.V. Chaudron, Combining metrics data and the structure of UML models using GIS visualization approaches, in: *Proceedings of the IEEE International Conference on Information Technology* 2005, vol. 2, 2005.
- [27] ISO/IEC FCD 9126-1.2, *Information Technology—Software Product Quality*, part i: quality model ed., 1998.
- [28] K. Khosravi, Y.-G. Guéhéneuc, Open issues with quality models, in: *Proceedings of the 9th QAOOSE workshop (ECOOP)*, 2005.

- [29] C.F.J. Lange, M.R.V. Chaudron, Managing model quality in UML-based software development, in: Proceedings of 13th IEEE International Workshop on Software Engineering and Practice (STEP'05), 2005.
- [30] V.R. Basili, G. Caldiera, H.D. Rombach, The goal question metric paradigm, in: Encyclopedia of Software Engineering, Wiley-Interscience, Hoboken, NJ, 1994.
- [31] C.F.J. Lange, Experiment replication package (<http://www.win.tue.nl/~clange>).
- [32] J. Cohen, A power primer, *Psychological Bulletin* 112 (1) (1992) 155–159.
- [33] Meerling, *Methoden en technieken van psychologisch onderzoek*, vol. 2, Boom, The Netherlands, 1989.
- [34] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslen, *Experimentation in Software Engineering—An Introduction*, Kluwer Academic Publishers, Dordrecht, 2000.
- [35] D. Lending, N.L. Chervany, The use of CASE tools, in: Proceedings of SIGCPR '98, ACM Press, New York, 1998.
- [36] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K.E. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, *IEEE Transactions on Software Engineering* 28 (8) (2002) 721–734.
- [37] M.J. Pacione, M. Roper, M. Wood, A novel software visualisation model to support software comprehension, in: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE), IEEE Computer Society Press, Silver Spring, MD, 2004.
- [38] C.F.J. Lange, Replication package of the metricview evolution experiment (<http://www.win.tue.nl/~clange>).