

# MetricViewEvolution: UML-based Views for Monitoring Model Evolution and Quality

Christian F.J. Lange, Martijn A.M. Wijns, Michel R.V. Chaudron  
Technische Universiteit Eindhoven, The Netherlands  
{C.F.J.Lange,M.R.V.Chaudron}@tue.nl, MartijnWijns@gmail.com

## Abstract

*As the role of models during software development and maintenance is becoming more and more important, techniques are needed to control a model's quality during evolution. We present our tool MetricViewEvolution as a step towards managing model quality during development and evolution. Six views are implemented in MetricViewEvolution that aid the user in tasks such as model understanding, identification of quality problems and evolution trends. The views combine structural model information with metrics data from inside the model and external sources. MetricViewEvolution has been applied successfully in industrial case studies.*

## 1 Introduction

The UML was originally intended to support designing software. Nowadays it experiences a much broader use. UML is also used to understand systems during maintenance, as a starting point for testing activities, for predictions and other uses. To perform these activities successfully, a developer needs several kinds of information such as metrics data, relations between model elements, information about non-functional quality attributes, and data about the model's evolution. The required information is available in different information sources such as UML case tools, UML metrics tools, and CVS logs. As most existing UML diagrams were adapted from other software modeling languages, they do not support to visualize all of the required information.

## 2 MetricViewEvolution

The purpose of the MetricViewEvolution<sup>1</sup> tool is to provide the developer with views, that are based on

<sup>1</sup>[www.win.tue.nl/empanada](http://www.win.tue.nl/empanada)

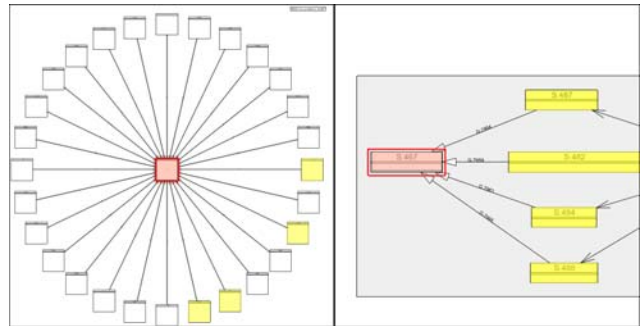
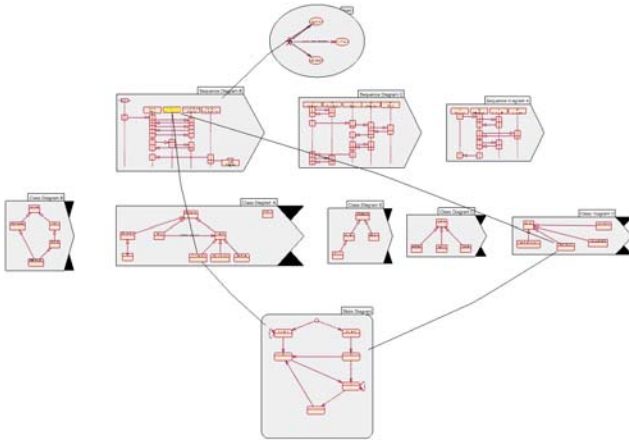


Figure 1. Context View (I.) vs. class diagram

the familiar UML diagrams, but that combine different types of information.

**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, such that it often occurs that only a limited context of model elements is viewed in one diagram. It is often tedious to explore the entire context of a class. The context view presents the entire context of a model element in one diagram and supports impact analysis and similar tasks. 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. An example is given in Figure 1.

**MetaView.** A problem that exists in regular UML 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. Figure 2 shows a MetaView in which inter-diagram relations are visualized.



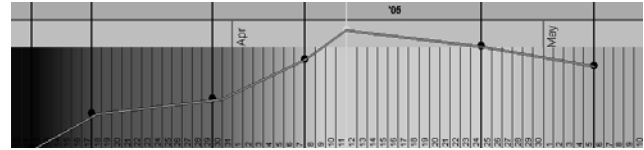
**Figure 2. MetaView supports Traceability**

**MetricView.** 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). Applying metrics to a UML model can result in an overwhelming amount of data. This data is 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 supports this problem by integrating the model and metric visualization. MetricView represents the metrics using color, size and/or shape to visualize values.

**UML-City View.** This view combines the concepts of the MetaView and MetricView. Metrics are visualized using 3D-boxes, mapped on top of the MetaView diagram. Low metric values are depicted by flat boxes while high values are depicted by tall boxes. Outlier values are identified easily.

**Quality Tree View.** Quality models such as provide a structure to decompose quality attributes such as maintainability and understandability into more concrete subconcepts. Each of these subconcepts can in turn be decomposed again. At the bottom of a quality tree are usually metrics. The Quality Tree View in MetricViewEvolution is a customizable quality tree that represents the values of quality attributes for UML models based on internal and external metrics. This functionality interacts with the built-in Evolution View to monitor the evolution of quality attributes over several versions.

**Evolution View.** Figure 3 shows the *evolution view*, in which the two concepts graph and calendar are combined to identify trends. The purpose of the *evolution view* is to enable users to spot trends in the values of quality attributes and/or metrics at multiple



**Figure 3. Evolution View.**

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.

A more elaborate discussion of the tool is given in [1].

### 3 Related Work

The SDMetrics tool [2] calculates metrics for UML models and presents the data in tables and graphs. Our MetricView represents metrics mapped on top of UML diagrams, whose layout reflects the developers mental map and, hence, is intuitively understandable. Lanza [3] proposes polymetric views to visualize properties. The main difference to our work is that we focus on UML-based visualizations targeted at various model-centric software engineering tasks. Hansen [4] reports positive feedback after using the MetricView within ABB. Several studies report about visualizing the evolution of source code, such as Voinea et al. [5] and Langelier and Sahroui [6]. However only little work addresses the evolution of UML models.

### References

- [1] C. F. J. Lange, M. A. M. Wijns, and M. R. V. Chaudron, "A visualization framework for task-oriented modeling using UML," in *Proc. of HICSS40*, 2007.
- [2] J. Wüst, "The software design metrics tool for the UML, version 1.3," <http://www.sdmetrics.com>.
- [3] M. Lanza, "Object-oriented reverse engineering – coarse-grained, fine-grained, and evolutionary software visualization," Ph.D. dissertation, Univ. Bern, 2003.
- [4] K. T. Hansen, "Project visualization for software," *IEEE Software*, vol. 23, no. 4, pp. 84–92, July 2006.
- [5] L. Voinea, A. Telea, and J. J. van Wijk, "CVSscan: Visualization of code evolution," in *SoftVis'05*, 2005.
- [6] G. Langelier and H. A. Sahraoui, "Animation coherence in representing software evolution," in *Proc. of (QA00SE'06)*, 2006, pp. 41–50.