Visual Exploration of Combined Architectural and Metric Information

Maurice Termeer Christian F. J. Lange Alexandru Telea Michel R. V. Chaudron

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
Den Dolech 2, 5600 MB Eindhoven, the Netherlands
m.a.termeer@student.tue.nl, c.f.j.lange@tue.nl, alext@win.tue.nl, m.r.v.chaudron@tue.nl

Abstract

We present MetricView, a software visualization and exploration tool that combines traditional UML diagram visualization with metric visualization in an effective way. MetricView is very easy and natural to use for software architects and developers yet offers a powerful set of mechanisms that allow fine customization of the visualizations for getting specific insights. We discuss several visual and architectural design choices which turned out to be important in the construction of MetricView, and illustrate our approach with several results using real-life datasets.

1. Introduction

UML diagrams are one of the most widespread forms of depicting software architectural and design information. UML models are usually created and used visually, using interactive modeling tools or diagram editors. Software metrics, such as produced by analysis tools [16], are efficient and effective instruments for analyzing large system architectures [3]. Metrics can answer complex, targeted questions, such as "which components are unstable or non-conforming to specific guidelines and requirements?" or "what happens if I change this component?" Metrics come mostly in two flavors. Global metrics, e.g. system cohesiveness or quality, characterize entire systems by single numbers, so they are best shown by tables with text and numbers. Per-element metrics characterize separate components or relationships, e.g. component coupling, fan-in, fan-out, 'provides', or 'uses'. To understand such metrics, tables are not enough. We need to correlate their values with already familiar, understood model information, such as contained in the various UML diagrams.

We present an approach that combines architectural and metric data on software systems in an integrated, interactive visualization tool called MetricView. We aim to create a single view where users smoothly and easily navigate between classical UML diagram data and architectural metric data, minimizing the cognitive disruption present in approaches that separate the two. Next, we let users easily, yet completely, customize the metric visualization in a variety of ways. Finally, we designed MetricView so that combining UML and metric data is easy and imposes no constraints or modifications on the data sources.

Section 2 presents related work on combining software metric and structural information. Section 3 details the visualization techniques we adapted and applied for our goals with MetricView. Throughout the presentation, we compare our experiences with MetricView and SoftVision [12], the latter being a related software visualization tool we developed in the past, and outline the lessons learnt. Section 4 concludes our discussion and outlines future work directions.

2. Related Work

We define the goal of software architecture visualization using the 5-dimensional model of Maletic *et al.* [9]: task, audience, target, medium, and representation. Our main **task** is to gain insight in the structure and semantics of architectures represented in the UML language. Our **audience** consists of system architects and developers, interested to understand a system's structure and dependencies, and evaluate various functional and non-functional component properties. Our visualization **target** is the system architectural information, given as a set of (class, sequence, package, etc) UML diagrams, enriched with various computed software metrics. The visualization **medium** is the standard PC display. Finally, the **representation** augments the classical UML diagram graphical layout used by modeling tools with metric data, shown as overlaid transparent icons.

UML-based modeling tools, such as Rational Rose [11] or Together [14], are the most accepted way for visually understanding architectures. However effective, such tools are limited to showing only UML diagrams. Adding extra information to the picture, e.g. software metrics, is not supported. At the other extreme, architectures can be analyzed by means of software metrics, computed by reverse engineering and software analysis tools and presented in tables and histograms [8][10]. This presentation form makes it hard to correlate metrics with structural information.

Somewhere between the above, programmable visualization tools such as Rigi [6], SHriMP [12] or SoftVision [13] propose a more abstract, system view which disposes of many rich UML visual details. Figure 1 (top) illustrates this in the SoftVision tool. Boxes are components, box nesting shows component inclusion (containment), and lines are component call relationships.

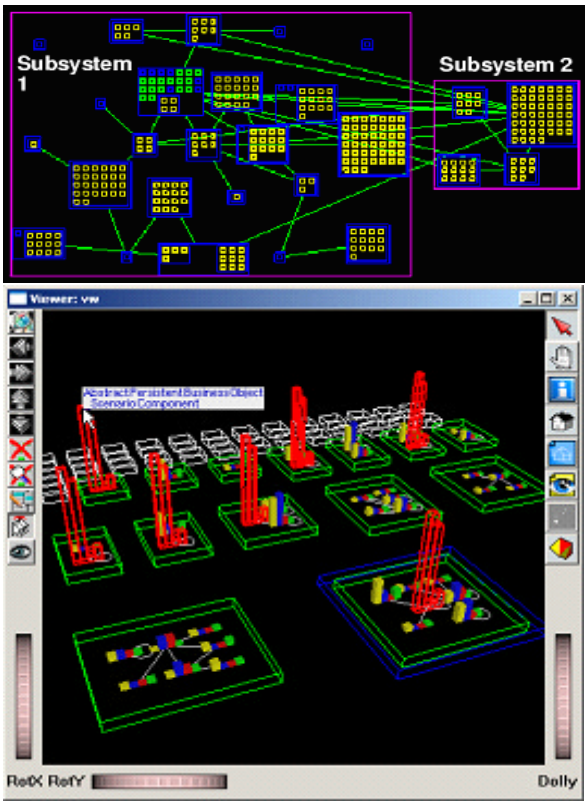


Figure 1: Software architecture without (top) and with metrics (bottom) in SoftVision

Being more customizable than the fine-tuned, but more rigid UML modelers, such tools allow users to *specify* several visualization elements. For example, software metrics can be displayed atop of the system structure graph, e.g. by tuning the color, shape, or size of the graph nodes to corresponding component metrics. Similar ideas have been presented in [1] and [8]. Figure 1 (bottom) shows a similar architecture as in Figure 1 (top). Each component has a four metrics bar chart laid out in the vertical dimension atop of the structure graph. Programming this visualization in SoftVision took us around two hours [15]. However useful, we discovered that this approach has several limitations. First, many users preferred the richer UML diagrams to our more simplified, albeit more customizable, visualization. Second, our users wanted a nearly automatic way to add metric visualization to their UML diagrams, in a single tool. We answered these requirements by combining the strengths of UML views (intuitive, interactive, visual navigation) and metric data (concise, precise) in an integrated tool, called MetricView. This tool is presented next.

3. Anatomy of MetricView

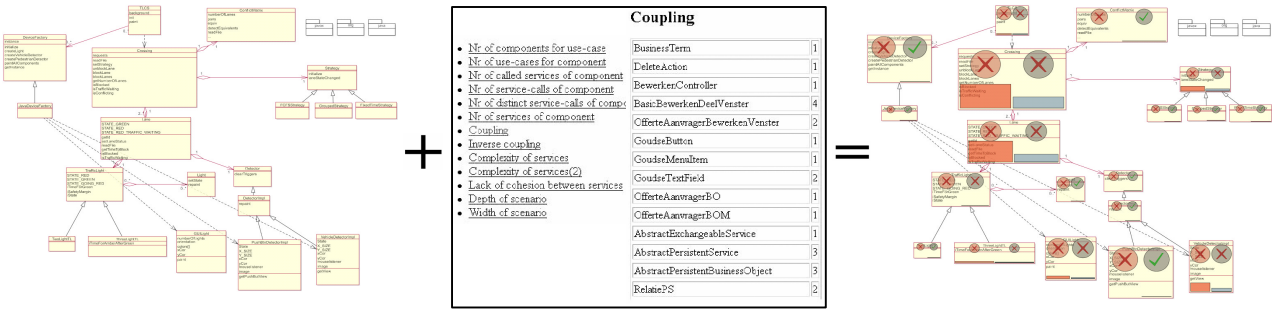
MetricView is essentially an UML visual tool that adds highly customizable metric visualizations to the well-known diagrams. In a nutshell, given a UML diagram (Figure 2a) and a set of metric values (Figure 2b), MetricView produces the result shown in Figure 2c. In the following, we describe the design (Section 3.1) and metric information (Section 3.2) used by MetricView. Next, we detail the visualization techniques we created to integrate the two in one view (Section 3.3).

3.1. Structure (UML) Data

MetricView can visualize class, sequence, state, use case, and collaboration UML diagrams, imported from XMI (XML Metadata Interchange) files conforming to OMG's version 1.2 [5]. The UML data is represented using the UML 1.3 metamodel [4]. Although these standards are a bit aged, they are still better accepted than their successors, XMI 2.0 and UML 2.0. At the time of writing, the UML 2.0 standard is still not yet released as final. Moreover, only very few UML tools support this format. Hence, our choice for the older and more supported format.

3.2. Metric Data

MetricView supports both *global* metrics, i.e., defined for a complete UML model, and *element* metrics, i.e., defined for an element, or relationship, of the model. A metric is modeled as a (key, value) pair. The key is the metric's (unique) textual name. MetricView currently supports boolean and numeric metrics. Any element can have any number of metrics. One may freely choose which metrics to define for which elements. Metrics and UML diagrams are provided as separate input files to MetricView. This loose association between the metric and structural data, similar to the one used by SoftVision [12], allows users to easily combine metric and UML data that come from independent tools. Indeed, our UML models came from various modelers [11][14]. So far, we used the over 40 metrics provided by our own software architecture analysis tool SAAT [10]. However, using metrics computed by other tools, e.g. [16], or alternatively UML models provided by different modelers, is clearly an easy task.



a) UML design information **b) Metric information** **c) Combination in MetricView**

Figure 2 : Combining UML design and software metric information in MetricView

3.3. Visualization

Figure 3 shows a typical visualization session in MetricView. The canvas (A) displays a UML class diagram, combined with six element (class) metrics. Users can select the desired diagram from the complete diagram (model) set from the XMI input file using the diagram browser (B). The UML diagram is drawn using the structural and layout information stored in the XMI input file. Layout data is, however, not a mandatory part of the XMI specification. In practice, different UML modelers may store different amounts of layout data, ranging from simple per-element 2D bounding box and position data to detailed geometry. MetricView is capable of drawing the UML diagrams even if only basic bounding box data are available, by performing a number of local element layouts using various graph layout techniques. The metric list (C) shows a textual list of all available metrics in the input file. In itself, this panel is similar to the text-based output of metric tools such as SAAT [10]. For every metric, the list shows its name, type (indicated by the letter "**b**" for boolean and "**i**" for integer), and a checkbox to select the metric for display (Figure 4 left).

Visualizing a metric proceeds as follows. First, the desired metric is checked in the list (D). A *metric icon* appears now atop of all UML elements in the canvas for which that metric is available. Several types of metric icons are available to choose from. They differ in the way they map the metric value to a visual attribute, as well as whether they work for boolean or integer metrics. We implemented the following integer metric icons (the visual attribute that maps the metric value is given in brackets): 2D rectangles (color, using a blue-to-red rainbow colormap), 2D height bars (*y* dimension), 2D circles (radius), 2D pies (circle arc), 3D bars (*z* dimension) and 3D cylinders (*z* dimension). For boolean metrics, we implemented several flavors of 2D checkbox icons. If several metric values are to be displayed for a UML element, MetricView lays out their chosen metric icons in a 2D grid layout over the element drawing itself. Finally, various metric icon specific parameters, such as cylinder icon and circle arc icon resolution, checkbox symbols, colormap color entries, and so on, can be tuned via GUI controls (E).

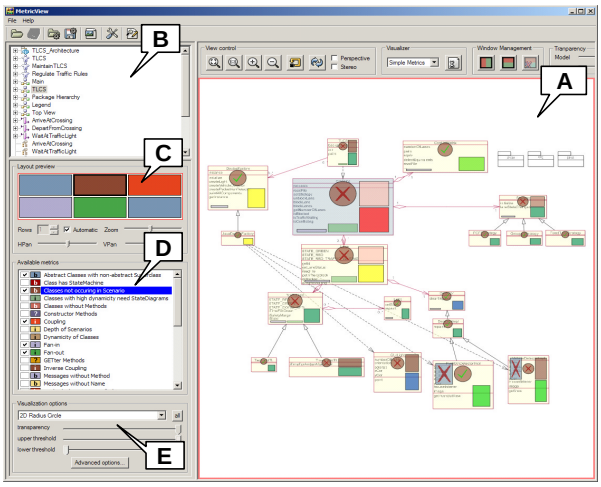


Figure 3: MetricView visualization overview

Figure 4 (right) shows such an UML class element with four metrics M_1, M_2 (boolean) and M_3, M_4 (integer) displayed, using two checkbox icon flavors and twice the same 2D height bar icon respectively. To let users make the correspondence between the displayed icons and the metrics in the metric list, we use two visual curs, as follows. First, a *layout legend* panel is drawn in MetricView (Figure 3C). The panel shows the grid layout used for to position icons over the UML elements in the canvas. Second, every metric in the metric list (Figure 3D) displays a small colored type symbol right to its check box (Figure 4 left). The layout legend displays the colors of the metrics that are selected from the metric list to be visualized in the canvas. Thus, the user can, in two steps, see which metrics are displayed over a given UML element, by a) looking at the color of the corresponding position in the layout legend and b) looking at the metric with that color in the metric list. Although direct icon-to-metric

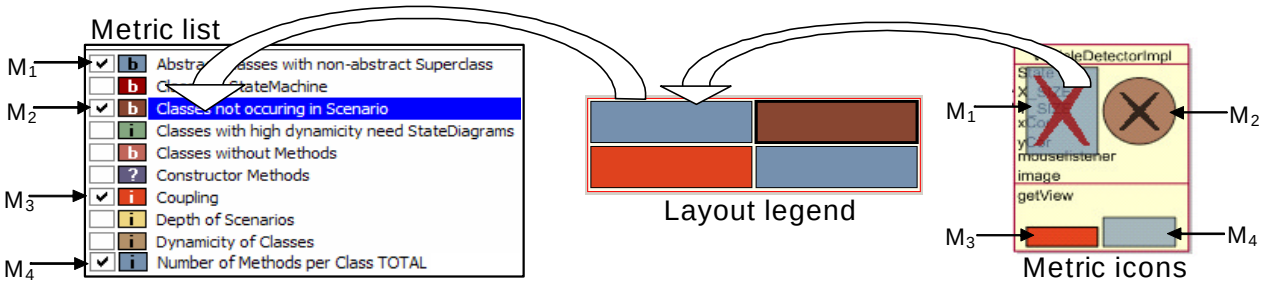


Figure 4: Visual mapping of metric list (left) to metric icons (right) via layout legend (middle)

association is also possible by clicking a metric icon in the canvas and getting its associated metric entry in the list, the previous two-step visual mechanism is better, since it allows one to directly interpret all metric icons present on all the canvas elements.

In comparison, SoftVision's icon customization features are technically more powerful than those of MetricView. SoftVision icons (called glyphs) can be any 2D or 3D graphical object, of which all attributes (shape, color, texture, lighting, size, and even interactive behavior) can be parameterized by any number of metric values by user-defined scripts. MetricView icons are a limited set of shapes, and the metric to shape attribute parameterization is strictly one to one. However, our extensive experience with SoftVision showed its icon mechanism to be often unnecessarily complex and hard to grasp for end users. Often, users want just a small icon type set, with straightforward parameterization and meaning, which is usable via pointing and clicking, with no scripting involved. Hence, our choice for the icon design used in MetricView.

A second visualization issue is how to let users freely navigate between the structural (UML) information and the metric information in the same view. We solved this problem by controlling the transparency of the two. By changing both the UML diagram (α_S) and metric icon (α_M) transparencies interactively via two sliders, users can effectively and efficiently change the focus from the structure (Figure 6 top, α_S=0.8, α_M=0.2) and metrics (Figure 6 bottom, α_S=0.2, α_M=0.8). In the extreme cases, we obtain a pure UML diagram visualization (α_S=1, α_M=0) or a pure histogram-like metric visualization (α_S=0, α_M=1).

A third visualization issue is the use of spatial dimensions. MetricView is able to do both 2D and 3D visualizations. Figure 5 (bottom) shows a 3D visualization, where the *xy* plane contains the UML diagram and the *z* dimension is used for the 3D metric icons. Although this visualization uses the same mechanisms as the one in Figure 1 (bottom) made with the SoftVision tool, the one made with MetricView provides more insight, due to the fine UML diagram detail available as well as the various navigation and metric customizations provided. Figure 5 (top) shows the same data as in Figure 5 (bottom), but using a 2D visualization. Interestingly enough, although we tried to provide well-tuned, advanced 3D support in MetricView, including 3D stereo display, most users preferred the 2D mode. We recorded the same experience from our use of SoftVision for software visualization in reverse engineering activities [12]. The only case, in both MetricView and SoftVision, when the use of 3D was preferred, was when users wanted to quickly get a comparative overview of several metric values defined for many elements of a given architecture. Using height bars produced here landscape-like visualizations such as Figure 1 and Figure 5, which, when navigated, allowed users to immediately spot outlier values (e.g. maxima).

Tuning transparency, as described before, prevents UML diagram element occlusion by the metric icons. Still, this is not a solution when one desires to view both metric and structural data. We solve this by allowing users to tune the metric grid layout by scaling and translating the 2D layout area used, on every element, to display the metric icons. Figure 5 uses this technique to 'shift' the metric icons to the upper-right quarter of the elements, making the UML annotations (class and method names, etc) visible. Another visualization issue is how to address questions such as "spot all components having important properties". We assume these properties are described by specific metric values or value ranges. To allow easy spotting of such components, we provide several simple interval-based, slider-like, filtering mechanisms in MetricView's interface. These allow users to select which metric values, or ranges, to display. No icons are displayed for metric values outside the selection, so this immediately lets users spot those diagram elements that match their selection. We did not implement more sophisticated metric filtering. Our previous experience with this situation in SoftVision showed that the best result is reached by computing more involved filtering as metrics and doing only basic filtering interactively.

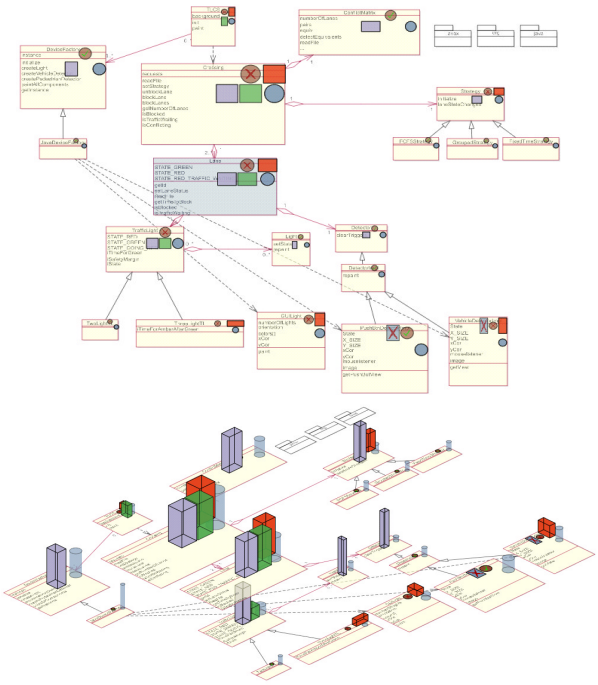


Figure 5: Planar (top) and 3D (bottom) layouts

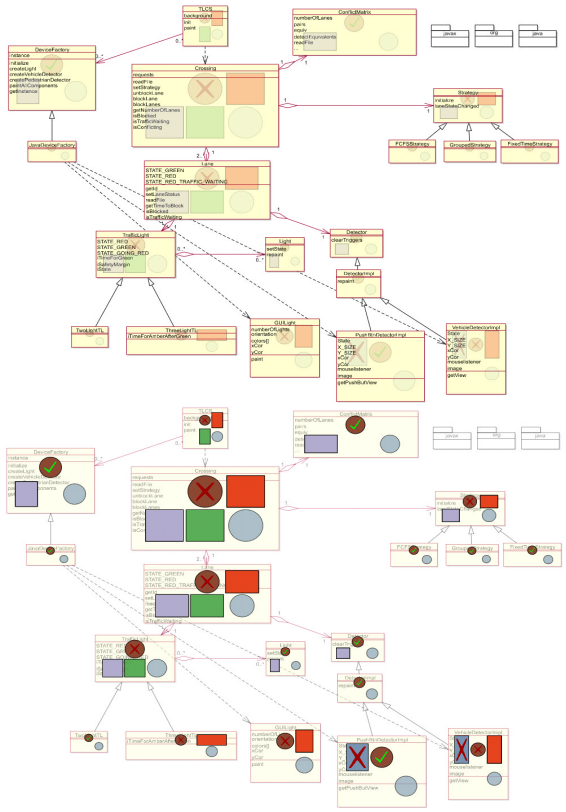


Figure 6: Tuning diagram and metric opacity

As a last example to illustrate the combination of structural and metric information, we present a visualization showing 15 metrics per diagram element (Figure 7). We use here the perspective, instead of the orthogonal, projection (compare to Figure 5 bottom). Although the displayed metric data amount per element is high, the 3D layout (*xy* plane for structure, *z* axis for metrics), and the usage of the same color for the same metric icon, provides an effective way to compare the various model elements.

MetricView is implemented in C++ using OpenGL for graphics, FreeType for the UML diagram high-quality fonts, and wxWindows for the user interface, and runs under both Windows and Linux. It can interactively visualize XMI datasets of tens of megabytes containing UML models up to thousand classes, on a Pentium 4 PC at 1.8 GHz with accelerated OpenGL. A prototype of MetricView showing all features presented in this paper, including an easy-to-use installer and example UML and metric data is publicly available at:

http://www.win.tue.nl/empanada/metricview

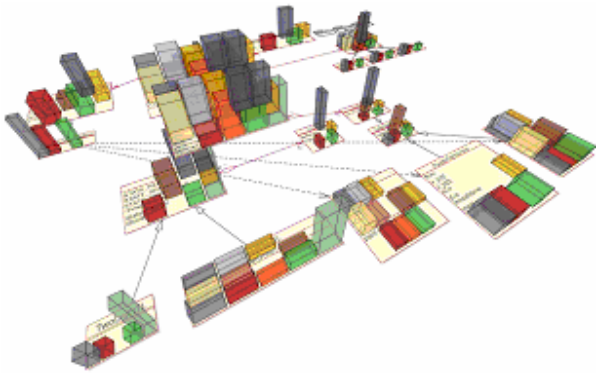


Figure 7: 3D perspective visualization with 15 metrics per component

4. Conclusions and future work

We have presented MetricView, an integrated software tool for interactive exploration of UML software models and software metrics. Throughout the design of MetricView, its users, and their preferences, stood central, as follows. First, MetricView builds upon the UML visualizations, using diagrams and graphical layouts which are familiar to software architects and developers. Metric information, computed by separate software architecture analysis tools, is added to the UML diagram visualization in a non-intrusive way. Users can continuously change the appearance of the visualization between the two extremes of a classical, architecture-only UML

diagram, and a histogram-like, metric-only display, by the simple dragging of a slider. Second, MetricView offers a wide range of fine-grained visualization customization options, that allow users to specify which metrics to display, how to arrange (layout) them, which graphical shapes, colors, sizes, and so on, to use for the metrics. Third, MetricView is designed to fully decouple the implementation details of its four main ingredients, or information types: the UML layout and structural information; the metric information; the metric layout (*where* to draw metrics); and the metric mapping (*how* to draw metrics). This allowed us, as proved by several use cases, to quickly build visualization scenarios that import UML information from various sources, e.g. modeling tools; add metric data computed with third-party software analysis tools; and easily choose, at run-time, which metrics to display, and how. Compared to our previous experience with SoftVision, which was designed for similar goals, MetricView allowed our users to combine structural and metric information in visualizations in a fraction of the time needed before, and with definitely more satisfying results. MetricView is an evolving project. We are currently working on several extension directions, as follows. First, we plan to integrate several graphical layout plug-ins, based on existing work in this area [1]. This will allow users to quickly produce quality visualizations even when no layout information is present in the UML input data, and also work on novel layouts to allow visualizing hundreds of elements on a single screen with minimal cluttering. Second, we plan to extend the metric visualizations beyond the metric-per-component current capabilities, e.g. by computing displaying more global, per subsystem, or per project metrics. Finally, we work on improving the metric computation tools themselves to extract more insightful and usable information from software architectures and display it within our improved MetricView tool.

References

[1] Diehl, S. (ed.), *Software Visualization*, Proc. Dagstuhl 2001 Intl. Seminar, Springer, 2002.

[2] Eiglsperger, M., Kaufmann, M., Siebenhaller M. A. *Topology-Shape-Metrics Approach for the Automatic Layout of UML Class Diagrams*, Proc. ACM SoftVis, ACM Press, 2003, pp. 189 – 198

[3] Fenton, N., Pfleeger, S. *Software Metrics: A Rigorous and Practical Approach*, Intl. Thomson Computer Press, 1996

[4] Fowler, M. *UML Distilled*, 3rd ed., Addison-Wesley, 2003

[5] Grose, T., Doney, G., Brodsky, B., *Mastering XMI: Java Programming with XMI, XML, and UML*, John Wiley & Sons, OMG Press, 2002

[6] Tilley, S. R., K. Wong, M. Storey, H. A. Müller, *Programmable Reverse Engineering*, Intl. Journal of Software Engineering and Knowledge Engineering, vol. 4, no. 4, World Scientific, 1994, pp. 501-520

[7] Maletic, J.I., Marcus, A., Collard, M.L. *A Task Oriented View of Software Visualization*, Proc. Vissoft'02, IEEE CS Press, pp. 32-40

[8] Lange, C., Chaudron, M., *Combining metric data and the structure of UML models using GIS visualization approaches*, Proc. Intl. Conf. on Information Technology: Coding and Computing, 2005, pp. 322 – 326

[9] Marcus, A., Feng, L., Maletic, J.I., *3D Representations for Software Visualization*, Proc. ACM SoftVis '03, ACM Press, 2003, pp. 27 – 36.

[10] Lange, C. F. J., *Empirical Investigations in Software Architecture Completeness*, Master's Thesis, Eindhoven University of Technology Press, 2002

[11] Rational Rose: www.306.ibm.com/software/rational/

[12] Storey, M.A., Best, C., Michaud, J., Rayside, D., Litoiu, M., Musen, M., *SHriMP Views: An Interactive Environment for Information Visualization and Navigation*, Proc. CHI '02, ACM Press, NY, 520 – 521

[13] Telea, A., Maccari, A., Riva, C., *An Open Toolkit for Prototyping Reverse Engineering Visualization*, Proc. IEEE VisSym '02, EG Association, 2002, pp. 241 – 251

[14] Together: http://www.borland.com/together, 2005

[15] Voinea, L., Telea, A., *A Framework for Interactive Visualization of Component-Based Software*, Proc. EUROMICRO '04, IEEE CS Press, 2004, pp. 567 – 574

[16] Wust, J. *SDMetrics: The software design metrics tool for UML*, http://www.sdmetrics.com, 2005