Visualizing Multivariate Attributes on Software Diagrams

Heorhiy Byelas and Alexandru Telea
Institute of Mathematics and Computer Science,
University of Groningen, the Netherlands
h.v.byelas@rug.nl, a.c.telea@rug.nl

Abstract

Software architecture diagrams and metrics are well-known and heavily used in many areas in software engineering. However, they are rarely combined in one (visual) representation. Although there are some advances in this direction, there are also some limitations. In this research, we study how to overcome these limitations. Specifically, we are interested in visualizing metrics on several levels of detail (classes, methods, groups of classes) on UML-like diagrams in a scalable and intuitive way. We present the results obtained in the first three years of the PhD track and outline ongoing work.

1. Introduction

Software diagrams show different types of software elements, *e.g.* interfaces, components, objects or roles, and structural or functional relationships in a system, *i.e.* the system structure. For object-oriented systems, UML diagrams are a conventional choice to represent the system structure [9]. Complementing diagrams, software attributes, encoded by software metrics, convey insights in system properties such as quality, maintainability, and performance. In such activities as forward and reverse engineering, reengineering, and maintenance, software architects need to easily correlate several metrics with the system structure.

Combining metrics and diagram information in a single representation is an effective way to help several types of system assessments, such as spotting (cor)relations among code attributes, relations, and diagram element types; determining where, on a system's architecture, do attributes reach outlier values; and identifying specific patterns and their correlation with metrics.

In our work, we explore and assess new ways to visualize attributes, modeled as a multivariate dataset, together with a system's structure diagrams, extracted from the architecture model or system's source code.

2. Research question

First, we introduce our data model. As input information, we consider a set of diagrams, such as class, component, or sequence diagrams, related to one software system. We assume the layouts of these diagrams are fixed and given. We do not want to change a given layout to show attributes (discussed below), as this can destroy the user's 'mental map', a well known fact in information visualization. We consider two types of attributes: software *metrics* and so-called *areas of interest* (AOI) [2], which are groups of diagram elements. For example, all safety-related components in a system can be grouped in the 'safety' AOI. Metrics can be defined on different types of diagram elements, like classes, relations between classes, methods, and AOIs. For example, the elements in the 'safety' AOI can have a safety-level metric value.

Our main research question is: How to visualize the combination of structure and attributes in a single drawing, such that

1. several metrics on classes, methods, and AOIs are shown in the same time,
2. several possibly overlapping AOIs are shown in the same time,
3. we use, but not change, a given (UML) diagram layout,
4. the generated image is easy to understand,
5. the rendering is fast even on complex diagrams with many attributes.

Our final goal is to enable users to correlate diagram structure with attributes, and attributes with other attributes.

3. Related work

Several attemps have been made to combine metrics and diagrams. Lanza *et al.* [8] render object-oriented class software metrics by mapping them to the class size and/or color.

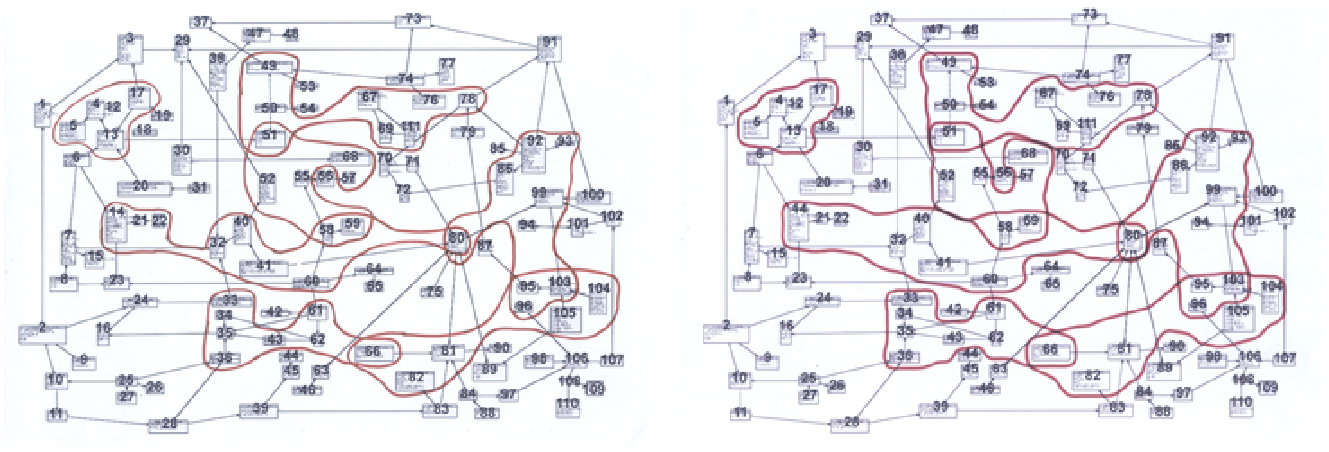

Figure 2. Human (left) and computer (right) drawn contours of the AOIs

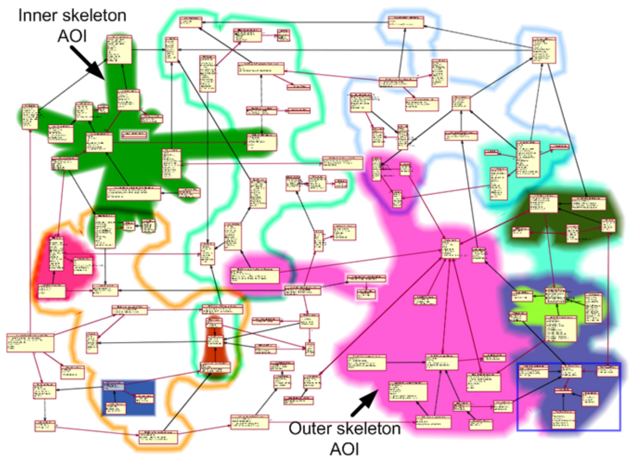

Figure 1. UML class diagram with mixed (inner and outer skeleton) AOI rendering

In this way, two metrics can be visualized simultaneously. A similar approach us used by the Rigi toolkit [11] and CodeCity [12], as well as many other software visualization tools [6]. However, the sizes of diagram elements, *e.g.* classes, may be constrained (fixed) to fit a preferred layout (see requirement 3 in Sec. 2), so they cannot be used to show a metric. Similarly, the class colors may be constrained *e.g.* in the case we like to draw the class method names on a fixed hue background. Termeer *et al.* [10] render class metrics by using icons scaled, colored, and drawn atop of the classes. However, this method does not scale well for method-level metrics and does not work for AOI metrics (see requirement 1 in Sec. 2). Moreover, the task of correlating *several* metrics on a *large* diagram is still an open problem.

4. Contributions

Our contribution to solving the research question stated in Sec. 2 is four-fold. We present two methods to draw complex, overlapping AOIs on large diagrams (Sec. 4.1). Next, we qualitatively and quantitatively compare our renderings with those produced by human users in an experimental study (Sec. 4.2). Third, we show how to render AOI metrics using a combination of shading and textures (Sec. 4.3). Finally, we show a scalable rendering technique for method metrics (Sec. 4.4).

4.1. Visualizing areas of interest

We propose two methods to draw AOIs: inner skeletons [1] and outer skeletons [2]. Both methods add the following requirements to those listed in Sec. 2:

- AOIs should be drawn with minimal visual clutter, even when they overlap,
- AOIs and diagrams should not visually interfere, *i.e.* should be drawn in different ways.

The inner skeleton technique constructs a star-shaped geometry connecting the AOI elements with their barycenter. In contrast, the outer skeleton technique incrementally deforms the AOI's convex hull to achieve a tight bounding shape, and also does additional processing to remove wrongly overlapping elements and improve the overall smoothness of the AOI. Figure 1 shows an example of a UML class diagram with more than 100 classes and more than 10 AOIs rendered with inner and outer skeletons.

We found that the outer skeleton method has less visual clutter and makes AOI overlaps easier to understand. Next, we qualitatively and quantitatively compared the results of this method to hand-drawn AOIs (Sec. 4.2).

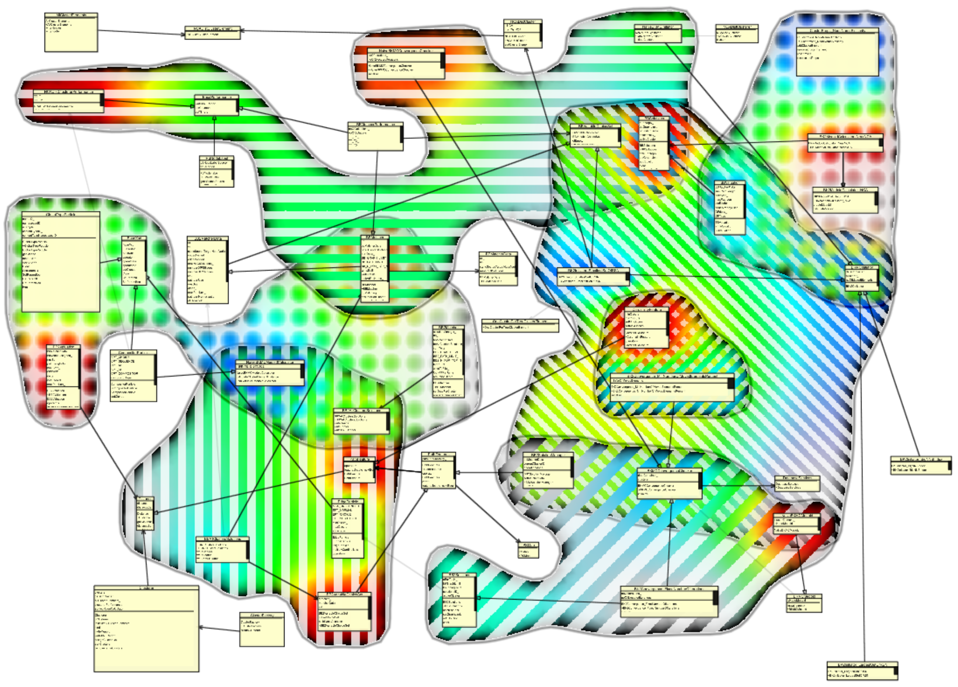

Figure 3. AOI metrics visualized with textures, shading and color interpolation

4.2. Evaluating the AOI visualization

We conducted a qualitative study that delivered insight in how users perceive the quality of computer-drawn AOIs [5]. The study involved thirty users of higher computer science education. The participants compared the computer-drawn AOIs with hand-drawn AOIs and set out technique drawbacks by completing a questionnaire. We used these results to further improve the outer skeleton rendering method (Sec. 4.1), by modifying our rendering method to imitate hand drawing aspects. Next, we designed a distance metric to quantitatively compare different AOI drawings, and used this metric to show that our improved rendering algorithm creates drawings which are closer to human drawings than our early version of the rendering algorithm.

Figure 2 shows a hand-drawn and a computer-drawn set of AOIs on the same diagram. As visible in this figure, our computer-drawn AOIs look natural and quite similar to the human-drawn ones.

4.3. Visualizing AOI metrics

After improving the geometry of AOIs, we developed a new rendering technique to show metrics values on them. Our new method simplifies the task of visually correlating the distribution and outlier metric values defined on AOIs with a system's structure [4]. As an additional requirement to those listed in Sections 2 and 4.1, we want to keep the elements' surfaces free to draw other information, such as method names or method metrics. Our solution combines texturing, blending, and smooth scattered data point interpolation (see Fig. 3). We show metric values using a red-to-blue colormap and use a set of pre-designed texture patterns to render several possibly overlaping AOIs. Our patterns are encoded in the texture transparency channel, so overlapping AOIs appear as woven patterns. Finally, we use shading to visually emphasize the borders of AOIs.

Our rendering technique can easily show 5 up to 10 AOIs, each with its own metric, on diagrams of tens of classes. The main limitation is the number of distinct AOIs that can overlap at one given place. Our technique gives good results for $n \leq 3$ AOIs overlapping in one place.

4.4. Visualizing method-level metrics

Finally, we visualize method metrics by adapting the well-known *table lens* technique to software diagrams [3]. As the basis of our new technique (we call it *metric lens*), we use a traditional UML class diagram, which displays all members within each class frame (see Fig. 4). Atop of this image, we display metrics following a table model, where the rows are methods and columns are metrics. Each table cell shows one metric. Metric values are rendered as scaled and colored pixel bars. In Fig. 4, we show two metrics: lines of code (left column) and McCabe's complexity (right column). Missing values have no bars. All metric tables can be

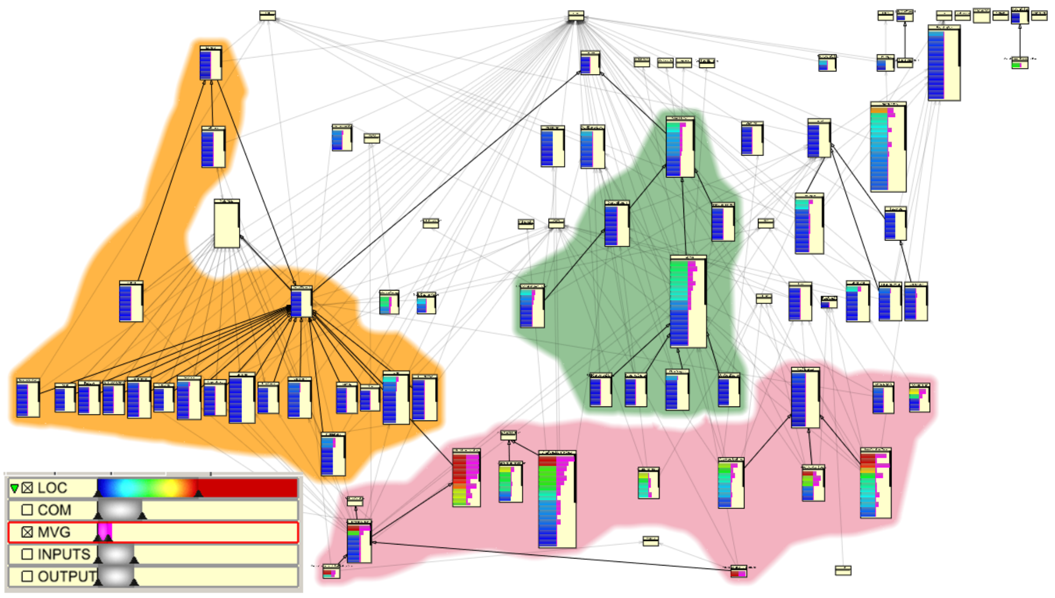

Figure 4. An example of a UML diagram with several AOI and method-level metrics

sorted on various criteria, like the metric values. This lets us easily locate outliers. There are cases when we do want to compare several logically related metrics. We provide interactive means to control in detail the visual mapping of metrics (visual ranges, colors, scales), thereby supporting several analysis scenarious.

The metric lens technique allows to display several tens of methods per class. The strongest scalability limitation regards the number of metrics that can be shown in the same time on a class. In our experiments, we saw that displaying more than three different metrics per method on large diagrams (100 classes or more) makes the result overcrowded and hard to read.

5. Conclusions and future work

Our visualization techniques allow to combine system's structure diagrams with software metrics defined on different levels, such as classes, methods and AOIs. We validated our methods by implementing them in a complete UML visualization tool, which was used by actual architects in the framework of a two-year industry-academic research project [7].

We plan to investigate how to display more metrics on the limited space offered by software diagrams, and how to visualize relations and metrics. Additionally, we would like to study how interaction can help understanding the metrics correlation and how to correlate metrics across the boundaries of single diagrams. Moreover, we plan to use our technique for visualizing multivariate metrics defined on 2D geographical data and general graph layouts.

References

[1] H. Byelas and A. Telea. Visualization of areas of interest in component-based architectures. In *EUROMICRO06*, 2006.

[2] H. Byelas and A. Telea. Visualization of areas of interest on software architecture diagrams. In *Proc. ACM SoftVis*, pages 105–114, 2006.

[3] H. Byelas and A. Telea. The metric lens: Visualizing metrics and structure on software diagrams. In *Proc. WCRE*, pages 339–340, 2008.

[4] H. Byelas and A. Telea. Texture-based visualization of metrics on software architectures. In *Proc. ACM SoftVis*, pages 205–206, 2008.

[5] H. Byelas and A. Telea. Towards visual realism in drawing areas of interest on software architecture diagrams. *J. of Visual Languages and Computing*, 2008. doi:10.1016/j.jvlc.2008.09.001.

[6] S. Diehl. *Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software*. Springer, 2007.

[7] ITEA. *Trust4All* project. 2006. `www.win.tue.nl/trust4all`.

[8] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006.

[9] OMG. *The Unified Modeling Language*. 2008. `http://www.uml.org`.

[10] M. Termeer, C. Lange, A. Telea, and M. Chaudron. Visual exploration of combined architectural and metric information. In *Proc. VISSOFT*, pages 21–26. IEEE Press, 2005.

[11] S. R. Tilley, K. Wong, M.-A. D. Storey, and H. A. Mller. Programmable reverse engineering. *Intl. J. of Software Eng. and Knowledge Eng.*, pages 501–520, 1994.

[12] R. Wettel and M. Lanza. Visual exploration of large-scale system evolution. In *Proc. WCRE*, pages 219–228, 2008.