The Metric Lens: Visualizing Metrics and Structure 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

We present the metric lens, a new visualization of method-level code metrics atop UML class diagrams, which allows performing metric-metric and metric-structure correlations on large diagrams. We demonstrate an interactive visualization tool in which users can quickly specify a wide palette of analyses, based on color-mapping, scaling, and sorting metric tables on UML diagrams. We illustrate our technique and tool by a sample complexity assessment analysis of a real-world C++ software system.

1. Introduction

In reverse engineering, one way to understand source code is to represent it on a higher abstraction level, *e.g.* as design or architectures. For object-oriented systems, UML diagrams are be a conventional choice to represent the system on a design level. Diagrams show the types of software elements and relationships in a system, *i.e.* the system structure. Software attributes, encoded by software metrics, convey complementary insights in system properties such as quality, maintainability, and modularity. Combining metrics and structure information in a single representation should be an effective way to help several types of system assessments, such as spotting (cor)relations among code attributes, relations, and diagram element types.

We propose here a technique that combines the *table lens*, a known technique for visualizing numerical tables, with UML diagrams, to scalably visualize code-level software metrics and structure. Section 2 presents our new structure-and-metric visualization and outlines its implementation. Section 3 shows a case study of our technique in understanding a real-world software system. Section 4 discusses the results and future work directions.

2. Metric Lens

As basis of our visualization, we use a traditional UML class diagram, which shows all textual methods[1] within each class frame (see Fig. 1). For each class C_i, we organize its method-level software metrics m_{ij}, extracted from code using one of the many available tools (*e.g.* Understand or SDmetrics), in a table structure T_i, with one row per method i and one column per metric j. Next, we render each table T_i atop of each class C_i in the diagram, using an adaption of the known table-lens visualization technique [2]. Missing metric values have no icon. The order of rows and columns in all metric tables of a diagram can be changed by various sorting criteria, such as method names or metric values, enabling different types of analyses.

Each table cell T_{ij} shows its metric value using a metric *icon*, *i.e.* a horizontal bar, scaled and/or colored by its metric value m_{ij}. The actual value-to-color or value-to-size mappings are fully user-controlled by a separate metric-lens *widget*, which acts also as a metric-value legend (Fig. 1 low-left). Next, we provide two independent zoom mechanisms, at *diagram* and *class* level. Diagram zooming allows users to focus on a specific subsystem. Class-level zooming, in combination with metric table sorting, is essentially the application of the classical table-lens principle within each class frame, and allows smoothly navigating between seeing an overview of each class as a set of colored bar graphs (when zoomed out), and seeing the individual member signatures (when zoomed in).

We implemented the metric-lens in a fully integrated reverse-engineering tool aimed at C++ code bases, atop the MetricView UML diagram tool [3]. Our tool includes software architecture extraction from source code and metric computation (not detailed here). All visualization is implemented using OpenGL. Additionally, our tool visualizes other architectural aspects using the *areas-of-interest* (AOI) technique [1].

3. Case study

To assess the effectiveness of our metric-lens visualization, we conducted a case study. Our question was: Could an investigator, not involved in a system's development but experienced in C++ and reverse-engineering, use our metric-lens visualization tool for a short period, to derive insight regarding the system's maintenance problems?

[1] Data members (also called data fields) are treated identically

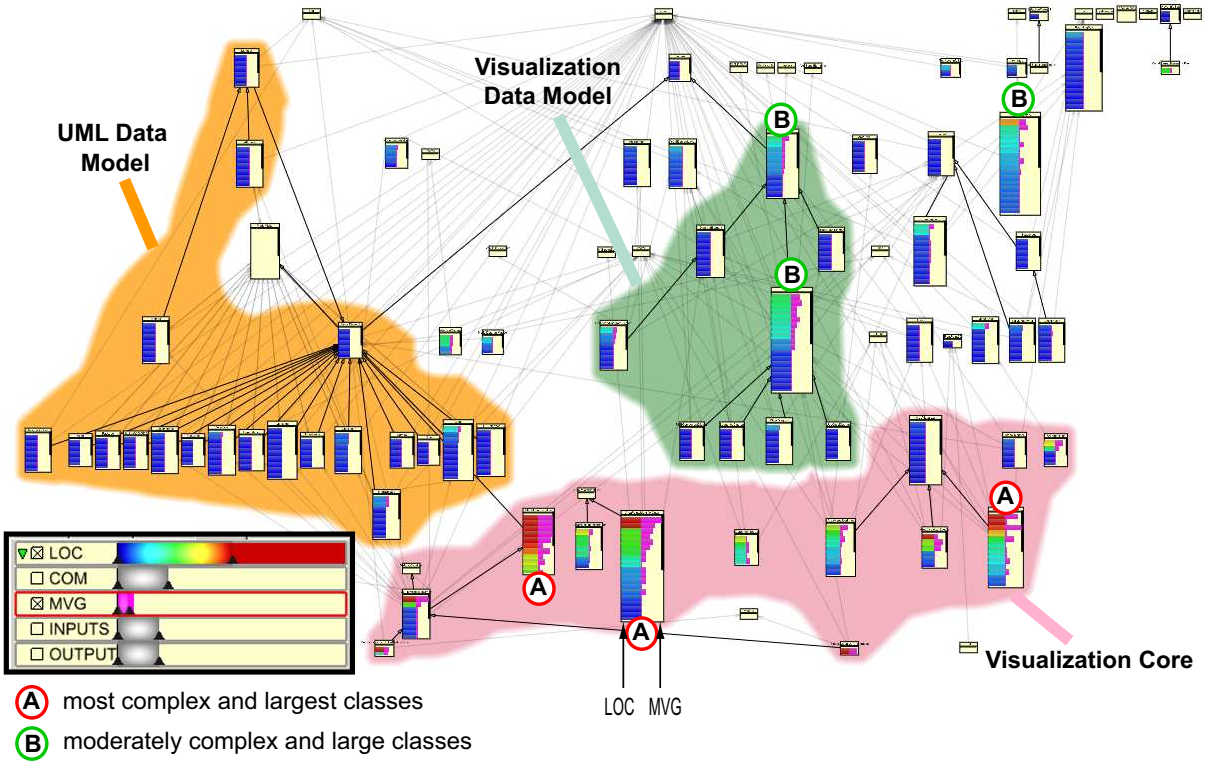

Figure 1. Complexity assessment of a UML diagram with three subsystems

In the example shown here, we analyze how complexity is spread over a C++ system containing about 15000 lines of code, in search for so-called complexity hot-spots, *i.e.* classes which may prove hard to understand or maintain. Figure 1 shows an extracted UML diagram with three subsystems (shown as colored areas-of-interest): UML Data Model, Visualization Data Model and Visualization Core (implementation). We computed the following metrics: lines-of-code (LOC) and McCabe's cyclomatic complexity (MVG) for each method. Using the metric-lens widget (shown lower-right), we mapped the LOC metric to rainbow-colormapped constant-size bars, the red color denoting values of 50 or larger. This is the left bar-graph in the classes in the figure. The MVG metric is visualized with purple bars scaled to the metric value, the longest bar denoting values of 10 or larger. This is the right bar-graph in the classes in the figure. Next, we sorted the metric tables decreasingly on LOC (from top to bottom of the class icons). We now quickly discover methods larger than 50 LOC and/or having a complexity above 10, which are figures that we consider to indicate a "complex" method, by looking for red, respectively long purple, bars.

We see that the Data Model classes are quite small and of low complexity, so Data Model is relatively simple and easy to maintain. In contrast, Visualization Core has large classes with large methods (warm colors in left bar braphs), and also the largest *and* most complex classe (marked A). This subsystem concentrates likely the highest complexity. Finally, Visualization Data Model contains small and low-to-medium complexity classes (*e.g.* the two marked B).

4. Conclusion and Future Work

During the validation phase, the developer confirmed the observations made, and conclusions drawn, by the investigator. Summarizing, we can say that combining the metric-lens technique (showing numerical values) with UML diagrams (showing system structure) effectively helps understanding the relations between metrics and structure at a finer level than diagrams themselves. We next plan to investigate how to display more metrics on the limited space offered by a class in a UML diagram, and also how to make the visual correlation of diagram relations and metrics more effective.

References

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

[2] R. Rao and S. Card. The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proc. CHI*, pages 222–230. ACM, 1994.

[3] 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.