CocoViz: Supported Cognitive Software Visualization

Sandro Boccuzzo and Harald C. Gall
Department of Informatics, University of Zurich, Switzerland
{boccuzzo, gall}@ifi.unizh.ch

Abstract

As software evolves and becomes more and more complex, program comprehension arises as a major concern in software projects. The amount of data and the complexity of relationships between the entities are unmanageable for engineers without effective tool support.

In this paper, we demonstrate how CocoViz[1] *can help understanding software in a quick and intuitive manner. Some of the implemented approaches have been presented independently before[1]. However, in CocoViz we combine them in an intuitive and easy to use manner.*

1 Introduction

The main contribution is an application we call CocoViz that software engineer take to inspect an evolving software system and find abnormalities. CocoViz uses cognitive shapes [1] compared to more abstract graphical representation used in approaches such as parallel coordinates [3]. Furthermore, we improve on the static polymetric views [4] with 1) an intuitive, interactive approach to the filtering of non relevant elements; 2) a normalization feature allowing the representation of well-designed classes as well-shaped cognitive metaphors (e.g., a well proportioned house), and therefore enabling comparability of projects through visualization even if the analyzed contextes differ substantially.

In the following, we describe our current implementation of CocoViz including the key visualization and the navigation concepts used to map software metrics to cognitive metaphors.

2 CocoViz Visualization

CocoViz implements concepts to improve the mapping of particular metrics to graphical elements following cognitive metaphors. Such metrics mapping has already been successfully applied in the RelVis approach [5] and in Polymetric Views by Lanza *et al.* [4].

[1]This work was partially supported by the Hasler Foundation (CH).

Dynamic Selection allows to limit the context of interest to certain software entities (*e.g.,* only potential God Classes - left of Fig. 1a). The context is dynamically arranged based on predicates that define the characteristics of the entities.

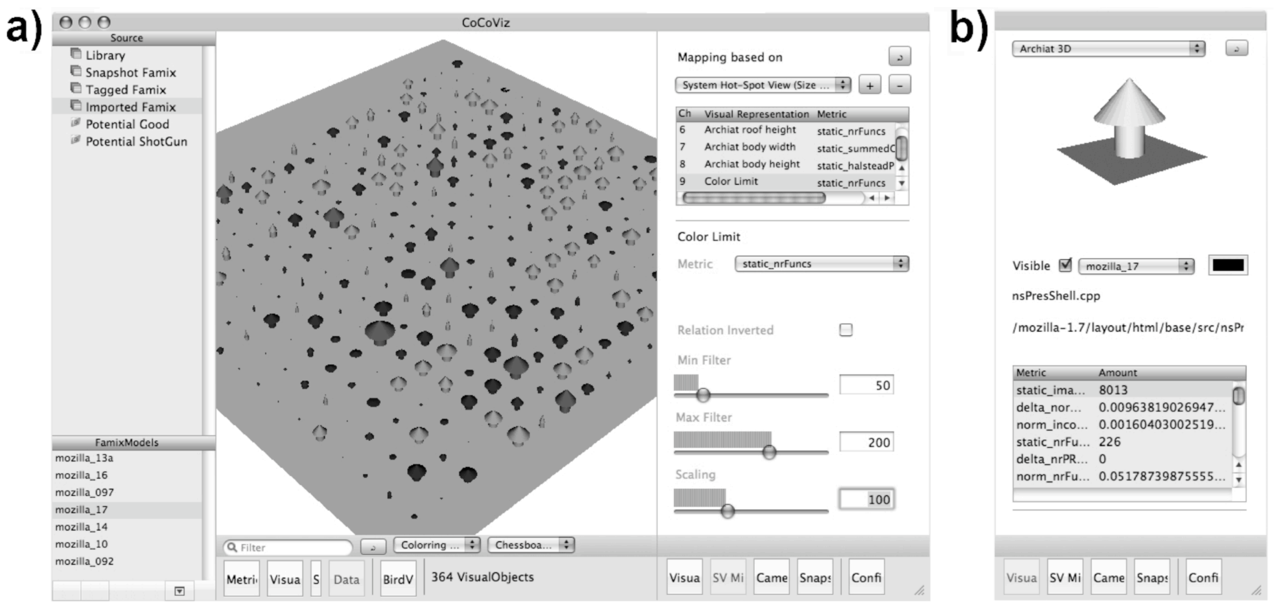
Metrics Clusters represent a set of metrics that together facilitate the visual interpretation of a software system. We implemented them as preset mappings within our Software Metrics Configurator. This offers an easy way to apply a metric cluster to different visualizations and at the same time allows a cluster to be enhanced or adjusted with other metrics to support additional analyses.

Software Metrics Configurator is used to deal with adequate metrics combinations and layouts. Similar to an audio mixers pitcher, where different audio sources are stretched or condensed to bring them in tune, we use the configurator to adapt the values of the mapped metrics to meet needs of a specific project. With such a configuration, well-shaped entities of the software are represented as a well-shaped glyphs. This concept also enables comparability of projects through visualization even if the analyzed context differ.

Software Visualization Mixer (SV Mixer) adopts concepts of an audio mixer for software visualization. Audio mixers process the level, tone, and dynamics of audio signals with equalizers, filters, limiters, and faders before sending the result to an amplifier. Our software visualization mixer processes the particular software metrics with filters, normalizers, and transformers before composing a visualization. The idea is to allow for the quick adjustment of the metrics to adapt the visualization for the analysis goal at hand.

To avoid information overload we use a simple filtering method to specify an interval of interest for every mapped metric (Fig. 1a bottom right). The sliders allow to quickly select the interval of interest *e.g.,* the highest or lowest 33% of elements. The threshold value for every mapped metric is currently calculated via linear regression (but not limited to).

Metaphors or Glyphs are visual representations of software metrics. They are generated out of a group of visual representations together with the corresponding metrics mapping. The mapped metric values specify the appearance of the glyphs. CocoViz uses simple and well-



Mapping based on
System Hot-Spot View (Size ...

Ch	Visual Representation	Metric
6	Archiat roof height	static_nrFuncs
7	Archiat body width	static_summedC
8	Archiat body height	static_halsteadP
9	Color Limit	static_nrFuncs

Color Limit
Metric static_nrFuncs
Relation Inverted ☐
Min Filter 50
Max Filter 200
Scaling 100

Visible ☑ mozilla_17
nsPresShell.cpp
/mozilla-1.7/layout/html/base/src/nsPr

Metric	Amount
static_ima...	8013
delta_nor...	0.00963819026947...
norm_inco...	0.00160403002519...
static_nrFu...	226
delta_nrPR...	0
norm_nrFu...	0.05178739875555...

Figure 1. a) CocoViz Window with Dynamic Selection, Visualization, SV-Mixer b) Element Inspector

known graphical elements familiar from daily life so that a viewer can quickly distinguish a well shaped glyph from a deformed one. (Fig. 1a shows the full Mozilla 1.7 open source project program code comprised of approx. 1.7 million lines of code with software entities drawn with the house metaphor - detailes of the case study can be found in [1]).

Tagging Glyphs and Visualization States in CocoViz implement concepts to preserve visualization states for later analysis. While navigating within a visualization, relevant aspects can be tagged. The remembered aspects can then later be analyzed and shared within a team. Tagging while dynamically interacting in a view allows one to store the interacting trail and therefore prevents engineers from getting lost within the visualization. Furthermore such a functionality allows one to use other visualizations and further examine the marked entities. Besides tagging, we can store the actual view in a snapshot.

Element Inspector gives detailed informations about the selected software entity (Fig. 1b for example, various metrics, such as complexity, lines of code, etc. and their evolution over time.

3 Conclusions

With CocoViz, we implemented a tool that supports the perception of relevant aspects in evolved software projects. The software visualization mixer, adapts the idea of an audio mixer to software visualization. With this mixer, metrics and their respective thresholds can be interactively configured. This allows to adapt the visualization to the analysis task at hand. Especially important is the capability to filter out irrelevant aspects to avoid information overload. The visualizations implemented by CocoViz are based on the concept of cognitive perceivable metaphors. A metaphor maps the metric values of the analyzed software to a visual representation, such as a house.

By defining a well-shaped object known to the observer, such as a well proportioned house, deviations from a sound design can be intuitively recognized. The configuration of such a reference object is done using the metrics congurator to map the software metrics to the attributes of the house, *e.g.,* lines of code to the height of the body and complexity to the width of the roof. With this normalized house in place, houses that are out of shape can be recognized easily and can then be further examined.

References

[1] S. Boccuzzo and H. C. Gall. Cocoviz: Towards cognitive software visualization. In *Proc. IEEE Int'l Workshop on Visualizing Softw. for Understanding and Analysis*, 2007.

[2] P. Dugerdil and S. Alam. Evospaces: 3d visualization of software architecture. In *Proc. IEEE Int'l Conf. on Softw. Eng. and Knowledge Eng.*, 2007.

[3] A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Proc. IEEE Conf. on Visualization*, pages 361–378, 1990.

[4] M. Lanza and S. Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *IEEE Trans. on Softw. Eng.*, 29(9):782–795, 2003.

[5] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *Proc. ACM Symp. on Softw. Visualization*, pages 67–75, 2005.