Evolution-Resilient Class Contours

Mattia Giannaccari[1], Marco Raglianti[1]

[1]REVEAL @ Software Institute – USI, Lugano, Switzerland

Abstract

Analyzing large scale object-oriented software systems is complicated. Analyzing their evolution increases the complexity by one order of magnitude, due to the additional dimension of *time*. While software visualizations can help to analyze a single system snapshot, having informative evolution-resilient visualizations is challenging.

We present our recent work on Class Contours, a novel visualization metaphor that depicts source code entities as *building facades* by mapping domain properties as, for example, code-level features (*e.g.,* attributes, methods) on visual properties (*e.g.,* doors, windows). Architectural patterns (*as in urban architecture*) emerge naturally. We explore an evolution of the current implementation of Class Contours to include time in a flexible yet deterministic, informative, robust, and scalable way.

Keywords

Class Contours, Evolution-Resilient Software Visualization, Software Evolution

1. Introduction

Comprehending classes is critical to evolve a codebase [1]. When analyzing object-oriented software systems, developers need to reconstruct the role and behavior of a class in their mental models, from scattered fragments of code [2], visualized as multiple continuous pages of text. Their focus often wanders from packages in a top-down approach to classes in a bottom-up fashion, alternating between different knowledge retrieval strategies in an opportunistic way [3]. Gathering new knowledge about the system incrementally, by looking at overviews, is complemented by inspecting key points in detail for specific application logic hotspots that provide information about the system's inner working.

To address these needs we proposed Class Contours [4], leveraging the *building facades* metaphor to use simple 2D architectural elements, which represent features of the classes in an intuitive and coherent way. In the resulting architectural patterns (as in *urban architecture*), similar class roles correspond to similar buildings in the Class Contours overview. It becomes easier at this point to spot and analyze similarities and differences, to identify outliers and application logic hubs, to allocate attention to specific parts of the system according to the task at hand. Internal (*e.g.,* attributes, methods) and external structure (*e.g.,* clients, providers) appear on the facade of the building. For example, the number of lines of code is mapped to the width of the structure, attributes are represented as doors, and methods as windows, while, at a glance, the repeating glyphs start to form a pattern language.

2. Evolving ZION

We implemented the Class Contours metaphor [4] in a visualization tool, ZION (for which a tool demo is under review at ICSE 2026), to validate our approach. With respect to the previous publication we already improved its parsing mechanisms to more reliably extract class features (and provide better future cross-language compatibility) by substituting VerveineJ with CodeQL.[1] Meanwhile, we started to consider strategies to compare class contours of two versions of a system to highlight the evolution direction and which domain changes trigger a visually recognizable structural change. To achieve this goal for evolutionary analysis, two features are missing from ZION's current implementation.

BENEVOL'25: Belgium Netherlands Software Evoltuion Workshop, November 17–18, 2025, Enschede, The Netherlands
mattia.giannaccari@usi.ch (M. Giannaccari); marco.raglianti@usi.ch (M. Raglianti)
https://mattiagiannaccari.github.io (M. Giannaccari); https://www.inf.usi.ch/phd/raglianti/ (M. Raglianti)
0009-0008-9356-2921 (M. Giannaccari); 0000-0002-6878-5604 (M. Raglianti)

[1]VerveineJ: https://modularmoose.org/developers/parsers/verveinej/ – CodeQL: https://codeql.github.com/

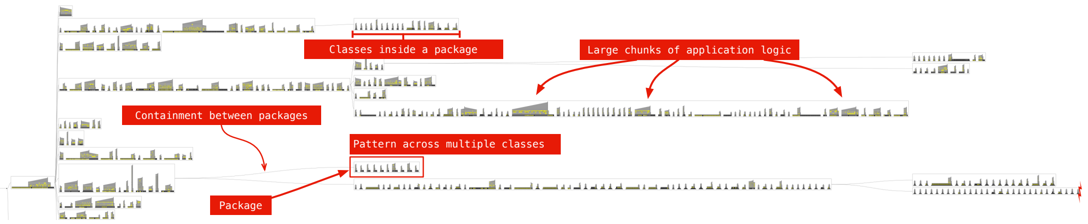
Classes inside a package
Large chunks of application logic
Containment between packages
Pattern across multiple classes
Package

Figure 1: Hierarchical view of Class Contours in a tree layout of packages for `antlr4` classes.

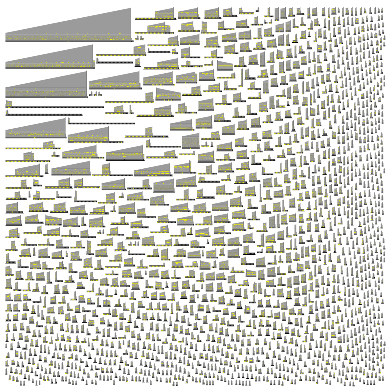

Figure 2: Class Contours rectangle packing (ArgoUML).

For positioning elements, ZION currently has just two layouts. A horizontal tree layout (Figure 1) to arrange Contours mirroring the structure of packages. And a rectangle packing layout (Figure 2) for space efficient placement in compact overviews, highlighting patterns at class level. We will discuss advanced potential layout strategies that remain consistent across different versions of the system, allowing to focus on the evolution of the contours themselves. While robust layouts that are evolution-resilient work for the city metaphor (*e.g.,* [5]), we would like to validate them in our simpler 2D representation.

The second missing feature is akin to normalization but involves deciding which metrics to visualize as *1-to-1 mapping* of the underlying feature and which are an *abstraction* of a core characteristic, independently from its "magnitude". The original intention of Class Contours was to represent the low level features in high fidelity when zooming in on a single building, while striking a convenient middle ground for scalability of overviews on large code bases. The new goal is to tackle the additional complexity of evolution, and time as a new dimension, while letting the Contours highlight important changes.

3. Conclusion

We present our current implementation of Class Contours, the recent update of the parser and its implications, while focusing on evolution-resilient layout strategies and normalization mechanisms to further extend the Class Contours beyond single snapshot analysis of a system. We sketch out for feedback the planned validation of our approach with the comparisons between UML class diagrams and class blueprints [6] on specific maintenance and evolution tasks.

Acknowledgments: This work is supported by the SNSF project "FORCE" (Project No. 232141).

Declaration on Generative AI: The author(s) have not employed any Generative AI tools.

References

[1] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Connallen, K. A. Houston, Object-Oriented Analysis and Design with Applications, 3rd ed., Addison Wesley, 2004.
[2] M.-A. Storey, Theories, methods and tools in program comprehension: Past, present and future, in: Proceedings of IWPC 2005, IEEE, 2005, pp. 181–191.
[3] M.-A. Storey, D. F. Fracchia, H. A. Müller, Cognitive design elements to support the construction of a mental model during software exploration, Journal of Systems and Software 44 (1999) 171–185.
[4] M. Giannaccari, M. Raglianti, M. Lanza, Skylines: Visualizing object-oriented software systems through Class Contours, in: Proceedings of VISSOFT 2025, IEEE, 2025, pp. 64–68.
[5] F. Pfahler, R. Minelli, C. Nagy, M. Lanza, Visualizing evolving software cities, in: Proceedings of VISSOFT 2020, IEEE, 2020, pp. 22–26.
[6] N. J. Agouf, S. Ducasse, A. Etien, M. Lanza, A new generation of CLASS BLUEPRINT, in: Proceedings of VISSOFT 2022, IEEE, 2022, pp. 29–39.