Visualization of Evolving Architecture Smells

Sandro Schulze
Anhalt University of Applied Sciences
sandro.schulze@hs-anhalt.de

Armin Prlja
TU Braunschweig
arminprlja@hotmail.de

Abstract

Architecture Smells (AS) have gained importance in recent past as indicator of bad practices related to the design of software systems. While AS and their symptoms show up on a rather abstract level compared to code smells, both share the characteristic that they evolve over time. This, in turn, may lead to even more severe smells that manifest themselves in the system. However, understanding the evolution of AS and when or how such smells tend to degrade in an undesirable way is not trivial given just tons of data from an analysis tool. In this paper, we introduce a visualization based on network graphs that support developers in understanding the evolution of three common architecture smells: cyclic dependencies, hub-like dependencies, and unstable dependencies. We also discuss scenarios when this is beneficial and what are current limitations of our visualization.
keywords software visualization, software evolution, architecture smells, software quality, software metrics

1 Introduction

As software evolves, it is likely that changes are made without adhering to design or coding principles, and thus, a software system exhibit *smells* causing quality degradation over time [4]. As a result, this may lead to undesired effects (e.g., reduced maintainability, increased bug proneness), commonly described as the metaphor of *technical debt* [3]. While there are different kinds of smells (e.g., code smells, test smells), in this work we focus on *architecture smells (AS)* that constitute violations against best practices for designing software architecture.

To gain comprehensive understanding of when and why architecture smells appear. As well as how they manifest in the system, the evolution of such smells is of superior interest [2, 5]. However, the plain data of such evolutionary analysis is of limited use for developers or software architect for understanding the origin and impact of architecture smells.

In this paper, we propose different visualizations that allow developers to investigate such AS in more depth and in an interactive way, and thus, allow to reason about birth, life, and death of such smells. While our visualization currently supports three common AS, namely cyclic dependencies (CD), hub-like dependencies (HD), and unstable dependencies (UD) [6], we only focus on CDs due to space limits.Particularly, we present the visualisations and briefly discuss why we have chosen them and which information they convey. Moreover, a preliminary study based on cognitive walkthrough indicates the usefulness of our visualization, even though a more profound empirical evaluation is required and planned for future.

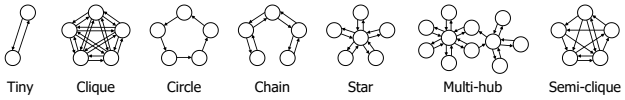

Figure 1: Different topologies for CDs based on Al-Mutawa et al. [1].

2 Background

We briefly introduce CDs and concepts that we developed to investigate their evolution [2].
Cyclic Dependencies exist between two or more components, if these components mutually depend on each other, and thus, increase the risk of ripple effects [6]. Such smells can exist on class-level as well as package-level (assuming Java and programming language). Moreover, CDs can differ in their shape, with more complex shapes indicating a higher negative impact on the overall system's quality [2]. In Figure 1, we we show the different shapes as proposed by Al-Mutawa et al. [1].

We introduced the notion of *intra-version* smells and *inter-version* smells to distinguish between a smell of one particular version and smells that exist over a certain period of time. More precisely, an intra-version smell is a smell in one particular version, whereas an inter-version smell is a set of related intra-version smells over multiple versions of a system.

Moreover, we consider *merging* and *splitting* in the evolution of CDs. In a nutshell, due to adding or removing dependency edges, CDs can merge together, resulting in a larger CD, or split, resulting in two smaller CDs. This concept allows us to perform novel analyses and identify possible situations that are critical wrt AS evolution.

3 Architecture Smell Visualization

For visualizing inter-version CD smells (see Figure 2 for an example), we have chosen a graph-based visual-

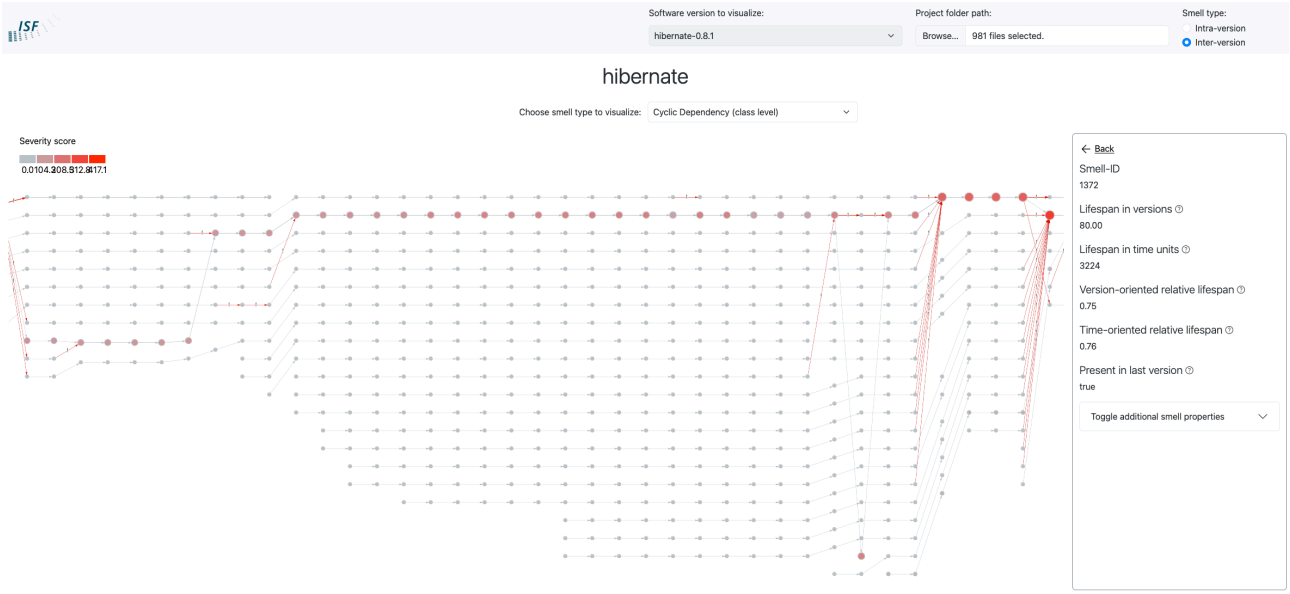

Figure 2: An example for an inter-version smell in hibernate with merging, splitting, and changes of shapes.

ization using network graphs with vertices representing an intra-version smell in a particular version and edges representing transitions between. This graph-based visualization allows us to visualize the different concepts such as merging/splitting, change of shapes, detailed information about participating intra-version smells, but also the time aspect to visualize the evolution over several versions. Next, we briefly explain how different kind of information is visualized.

Merging & Splitting. We use parallel (sub)graphs for indicating CD families that are subject to merging/splitting. In case that such CDs merge, multiple vertices exhibit an outgoing edge to the same successor vertex. In contrast, if a split occurs, this is visualized by one vertex from which multiple outgoing edges have different successor vertices.

Change of Shapes. During evolution, inter-version smells can change their shape, e.g., from a star-like shape to a multi-hub. In our visualization, we indicate such a change by a collared edge that additionally is annotated with an exclamation mark. When hovering over this exclamation mark, a tool tip provides information about the the source and the target shape.

Details of Intra-Version Smells. AS inter-version smells consist of a set of related intra-version smells, it may be of interest to inspect a particular intra-version smell. We support this kind of exploration; by just clicking on an intra-version smells, information shows up by means of metrics such as size, order, or severity score. As future work, we also want to allow users to seamlessly witch to the detailed visualization of the intra-version smell.

Metrics. Eventually, we provide various characteristics of inter-version smells either explicitly or implicitly to the user. For instance, the size of the vertices corresponds to the size of an intra-version smell ,that is, the number of classes/modules that form the intra-version smell of a particular version. Additionally, the color of a vertex corresponds to its severity score with a red note indicating a higher severity score. Finally, we provide additional metrics for the shown inter-version smell, such as lifetime, in a menu on the right-hand side.

References

[1] H. A. Al-Mutawa, J. Dietrich, S. Marsland, and C. McCartin. On the shape of circular dependencies in Java programs. In *Proceedings of the Australian Software Engineering Conference (ASWEC)*, pages 48–57. IEEE, 2014.

[2] P. Gnoyke, S. Schulze, and J. Krüger. An evolutionary analysis of software-architecture smells. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 413–424. IEEE, 2021.

[3] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21, 2012.

[4] D. L. Parnas. Software aging. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 279–287. IEEE, 1994.

[5] D. Sas, P. Avgeriou, and U. Uyumaz. On the evolution and impact of architectural smells: An industrial case study. *Empirical Software Engineering*, 27(4), 2022.

[6] G. Suryanarayana, G. Samarthyam, and T. Sharma. *Refactoring for Software Design Smells: Managing Technical Debt*. Elsevier, 1 edition, 2014.