How Deep is the Mud: Fathoming Architecture Technical Debt Using Designite

Tushar Sharma
Dept of Management Science and Technology
Athens University of Economics and Business
Athens, Greece. tushar@aueb.gr

***Abstract*—The quality of software architecture is an important concern for any software development team. Architecture smells represent quality issues at architecture granularity. Identifying and refactoring them periodically is a necessity to keep architecture quality high. We present *Designite*, a software design quality assessment tool, that identifies seven well-known architecture smells. Along with the identification, the tool provides supplementary information such as *cause* and *responsible classes* for each identified smell instance to help developers understand and refactor the smell. The tool is relevant and useful in both research and practice context. Software developers may use it to identify technical debt instances and to refactor them. On the other hand, software engineering researchers may use the tool to carry out large-scale empirical studies concerning code smells.**

Demo URL: https://youtu.be/ogrAxQLKwsU

***Index Terms*—Architecture smells, code smells, technical debt, maintainability, code quality, refactoring.**

I. Introduction

Presence of code smells [1] indicates quality issues in a software system. A large volume of smells in a software system leads to poor maintainability that in turn makes the evolution of the software difficult. Therefore, identifying smells and refactoring them periodically is a necessity to keep the technical debt [2] low.

Smells in production code are classified as implementation smells, design smells, and architecture smells based on their granularity, scope, and impact [1]. A software system's architecture represents the key design decisions spanning multiple components and having a system-level impact [3]. Therefore, the impact of architecture smells is system-wide.

Identification of smells is the first step towards gauging the technical debt accrued within a software system. Software engineering community has proposed many smell detection tools based on techniques such as metrics, heuristics, and machine-learning [1]. However, despite some attempts [4], [5] architecture smells detection is still not well supported by the existing tools.

Designite [6] is a software design quality assessment tool. Apart from supporting detection of a wide variety of design and implementation smells, it detects seven well-known architecture smells for C# code. Other key features supported by the tool are code metrics computation, dependency structure matrix, trend analysis of smells, code-clone detection, integration with external tools via its console application, and hotspot analysis. The tool provides interactive visualizations, such as sunburst and treemap, for the detected smells and metrics. Figure 1 shows the tool's presentation of identified smells using a sunburst diagram facilitating smells' filtering and navigation interactively. These visualization aids make it easier for the users to comprehend the results of the analysis. The tool offers free *academic* licenses for all academic purposes.

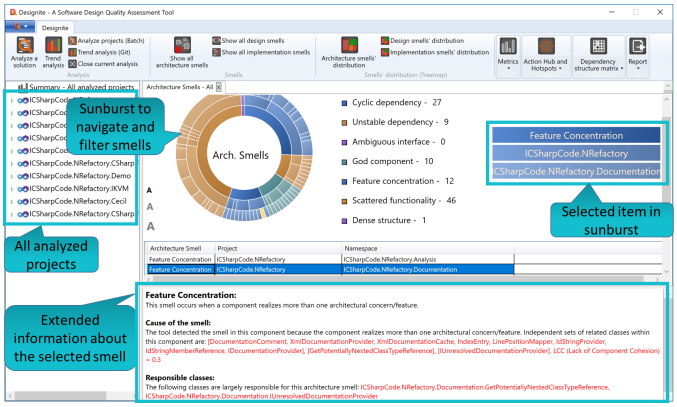

Fig. 1. Presentation of identified smells in Designite

II. Key Features

In this section, we elaborate on the features offered by the tool specific to architecture smells detection.

A. Architecture Smells Detection

We provide a brief description of the supported architecture smells below.

- **Cyclic Dependency:** This smell arises when two or more architecture components depend on each other directly or indirectly [7].
- **Unstable Dependency:** According to Stable Dependencies Principle (SDP) [9] the dependencies between packages should be realized in the direction of the stability of the packages. Hence, a package should only depend on packages that are more stable than it is. This smell occurs when a component depends on other less stable components [8].
- **Ambiguous Interface:** A component that offers only a single, general entry-point into the component suffers from this smell [10].

- **God Component:** This smell occurs when a component is excessively large either in terms of lines of code or number of classes [7].
- **Feature Concentration:** This smell occurs when a component is not cohesive *i.e.,* realizes more than one architectural concerns or features [11].
- **Scattered Functionality:** This smell arises when multiple components are responsible for realizing an architectural concern [10].
- **Dense Structure:** This smell arises when components exhibit excessive and dense dependencies without any particular structure [12].

B. Responsible Classes Identification

With each identified architecture smell, the tool shows *cause* as well as *responsible classes* for the smell instance. The cause summarizes rationale of the smell identification and the responsible classes show the classes that contribute non-trivially to the occurrence of the smell's instance. Identification of the responsible classes significantly help developers approach refactoring of these smells.

C. Tracking Smells

Identifying smells is only the first step towards better code quality; a follow-up action is required to reap the benefits of using a smell detection tool. In a real-world setting, upon detection, a developer may neglect the smell, may decide to discard it when she thinks the identified smell is a false-positive or may not be in position to refactor it (for instance, in case of legacy code), or may decide to refactor it (immediately or in future). Designite offers a dedicated feature *i.e., Action Hub* to track and manage identified smells and plan their refactorings. Action Hub provides a consolidated view of all the identified smells and their corresponding status (one of the following: *identified, drop, refactor*, or *wrong*). This is a pragmatic feature for the developers to know precisely the status of their code quality and associated actions.

D. Visualization

Visualization is one of the vital strengths of the tool. Apart from sunburst representation shown in Figure 1, the tool offers other visualization aids such as smells' treemap (to show distribution of smells across the project and to highlight hotspots) and metrics pie-chart.

E. Console Application

Designite offers a console application to perform code quality assessments and emit the output in a desired format (CSV, XML, or spreadsheet). The console application could be used in a continuous integration workflow to enforce code quality checks. Furthermore, the application is very relevant for large-scale empirical studies (such as one performed by Sharma et al. [13]) that involve mining software repositories to extract code smells.

III. Related Work

The software engineering community has put together a plethora of smell detection tools using various techniques. The prominent techniques used to detect smells are metrics-based, rules (or heuristics)-based, machine learning-based, and history-based [1]. However, the existing tools mainly focus on implementation and a few design smells. There has been some attempts to detect architecture smells. For instance, Titan tool-set [5] detects modularity violations such as cyclic dependencies. Similarly, Arcan [4] detects three architecture smells. Despite these attempts, the available tool support for architecture smells is very limited and supports only Java programming language. The proposed tool fills the gap of a missing architecture smells detection tool for C# code.

IV. Conclusions

Designite is a software design quality assessment tool for practitioners and researchers to reveal architecture, design, and implementation smells. The tool detects well-known architecture smells and presents them using visual aids to make it easier for developers to comprehend them. The tool offers supplementary information and features such as Action Hub to facilitate developers track, manage, and refactor smells and help them keep technical debt low.

References

[1] T. Sharma and D. Spinellis, "A survey on software smells," *Journal of Systems and Software*, vol. 138, pp. 158 – 173, 2018.

[2] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, 1st ed. Morgan Kaufmann, 2014.

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[4] F. A. Fontana, I. Pigazzini, R. Roveda, D. Tamburri, M. Zanoni, and E. D. Nitto, "Arcan: A Tool for Architectural Smells Detection," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, Jun. 2017, pp. 282–285.

[5] L. Xiao, Y. Cai, and R. Kazman, "Titan: a toolset that connects software architecture with quality analysis," in *FSE 2014: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Drexel University. ACM, Nov. 2014, pp. 763–766.

[6] T. Sharma, "Designite - A Software Design Quality Assessment Tool," May 2016, http://www.designite-tools.com. [Online]. Available: https://doi.org/10.5281/zenodo.2566832

[7] M. Lippert and S. Roock, *Refactoring in large software projects: performing complex restructurings successfully*. John Wiley & Sons, 2006.

[8] F. A. Fontana, J. Dietrich, B. Walter, A. Yamashita, and M. Zanoni, "Antipattern and Code Smell False Positives: Preliminary Conceptualization and Classification," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2016.

[9] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[10] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying Architectural Bad Smells," in *CSMR '09: Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, Mar. 2009, pp. 255–258.

[11] H. S. de Andrade, E. Almeida, and I. Crnkovic, "Architectural bad smells in software product lines: An exploratory study," in *Proceedings of the WICSA 2014 Companion Volume*. ACM, 2014.

[12] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in *Proceedings of the 13th International Workshop on Mining Software Repositories*, ser. MSR'16, 2016, pp. 189–200.

[13] ——, "House of Cards: Code Smells in Open-Source C# Repositories," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Nov 2017, pp. 424–429.