REVEALING DESIGN PROBLEMS IN STINKY CODE

This briefing reports scientific evidence on how to help developers to use code smells for revealing design problems in the source code.

FINDINGS

- The findings of this briefing regard the use of code smells for revealing design problems in the source code.

- A design problem is a design characteristic that negatively impacts quality attributes. Each design problem is often located in program locations, such as packages and hierarchies. The prevalence of design problems in a program may lead to severe consequences on software quality.

- Existing studies propose the analysis of code smells for assisting developers on revealing design problems. The analysis of code smells can be performed with the help of smell detection and visualization tools.

- Previous studies state that the stinkier a program location is, the more likely it contains a design problem. However, there is little knowledge if developers can effectively identify design problems in stinkier code.

- Through a mixed-method study, we asked professional developers to identify design problems through the analysis of agglomerated and non-agglomerated code smells.

- An agglomeration of code smells is a group of inter-related code smells occurring in the same program location.

- For example, an agglomeration of code smells may occur in a hierarchy of classes (Figure 1). Multiple classes of such a hierarchy are affected by code smells. To identify a design problem, developers may analyze and correlate the information provided by each code smell agglomerated in the hierarchy.

- The overall precision of developers using agglomerations was 29% higher than the precision of developers using non-agglomerated code smells. However, the analysis of multiple code smells and their relationships in the code is still challenging and time-consuming to most developers.

- Agglomerated code smells helped developers to reveal a few more design problems than non-agglomerated code smells. This small difference occurred because developers still spent considerable effort with the analysis of irrelevant code smells in an agglomeration.

- To help developers in revealing more design problems, the prioritization of agglomerations is required. Developers should focus on the most relevant agglomerations of code smells, discarding the irrelevant ones. However, there is no prioritization criteria that is effective for any system.

- Existing ranking criteria should prioritize those agglomerations that are *cohesive*. A cohesive agglomeration is an agglomeration in which all the code smells are related to the same, single design problem. Many agglomerations are not cohesive. Figure 2 shows an example of non-cohesive agglomeration.

- Agglomerations helped developers to avoid false positives when identifying design problems. Agglomerated code smells assisted developers to indicate less false positives than non-agglomerated code smells.

- For both agglomerated and non-agglomerated code smells, false positives occurred because developers were unable to perform deeper analysis of the stinky program locations.

- Usually, a developer needs to analyze multiple classes and methods to spot the full extension of a design problem. Agglomerations helped developers to perform such analysis. However, this analysis was much more difficult with the non-agglomerated code smells.

- To reduce false positives, developers need better support: (1) an improved visualization tailored for agglomerations, (2) on demand examples of design problems, and (3) tips on which design problem each agglomeration may indicate.

- The use of agglomerations for revealing design problems is promising. However, as presented in this briefing, there are many open challenges that should be addressed before agglomerations can be used in a time-effective manner.

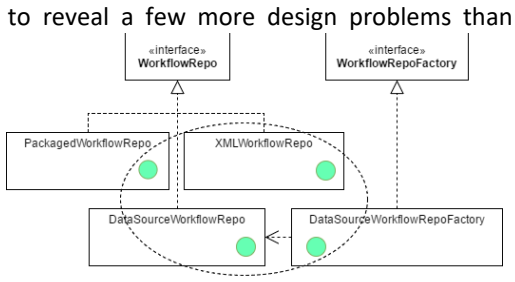
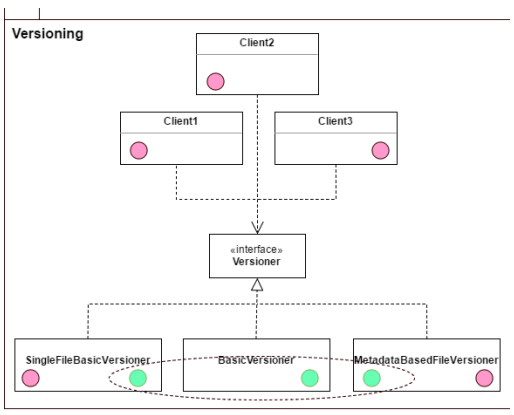


Figure 1: Example of Hierarchical Agglomeration



Keywords:

Design problem
Software Design
Code Smell
Agglomeration

Who is this briefing for?

Researchers and practitioners who want to understand how to use code smells for revealing design problems.

Where the findings come from?

A mixed-method study involving eleven professional software developers and two open source systems.

What is a mixed-method study?

A study performed with two or more empirical methods.

What is included in this briefing?

The main findings of the original mixed-method study, and brief information about the context of the findings.

What is not included in this briefing?

Descriptions about the research method or details about the results presented in the original paper.

For additional information about this research:

wnoizumi.github.io/SBCARS2017

OPUS Research Group:

http://les.inf.puc-rio.br/opus/

ORIGINAL RESEARCH REFERENCE

Oizumi et al. ***Revealing Design Problems in Stinky Code: A Mixed-Method Study***. Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS), 2017.