M. Tichy, E. Bodden, M. Kuhrmann, S. Wagner, J.-P. Steghöfer (Hrsg.): SE 2018,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2018 143

On Continuous Detection of Design Flaws in Evolving Object-Oriented Programs using Incremental Multi-Pattern Matching

Sven Peldszus[1], Géza Kulcsár[2], Malte Lochau[2], Sandro Schulze[3]

Abstract: This work has been initially presented at the International Conference on Automated Software Engineering (ASE) 2016 [Pe16]. Design flaws in object-oriented programs may seriously corrupt code quality thus increasing the risk for introducing subtle errors. Most recent approaches identify design flaws in an ad-hoc manner, either focusing on software metrics, locally restricted code smells, or on coarse-grained architectural anti-patterns. In our work, we utilize an abstract program model capturing high-level object-oriented code entities, further augmented with qualitative and quantitative design-related information. Based on this model, we propose a comprehensive methodology for specifying object-oriented design flaws by means of compound rules integrating code metrics, code smells and anti-patterns in a modular way. This approach allows for efficient, automated design-flaw detection, by facilitating systematic information reuse among multiple detection rules as well as between subsequent detection runs on continuously evolving programs.

Summary

Object-oriented programming offers software developers rich concepts for structuring initial program designs. As software systems tend to become more and more long-living, their initial code bases have to be continuously maintained, improved and extended. In practice, corresponding evolution steps are frequently conducted in an ad-hoc manner. As a result, the initial program design may be prone to continuous erosion, eventually leading to structural decay whith negative side-effects such as increasing the risk for introducing subtle errors.

Object-oriented refactorings have been proposed as effective counter-measure against design flaws. In fact, a manual identification of problematic code structures to be removed by applying appropriate refactorings is tedious, error-prone, or even impossible for larger-scaled software projects. Various approaches have been recently proposed to assist and/or automate the identification of design flaws. The different attempts may be roughly categorized into three kinds of *symptoms*, potentially indicating object-oriented design flaws [Mo10].

- *Software metrics* assess quality problems in program designs by means of quantified measures on structural code entities (e.g. low cohesion of classes).

[1] University of Koblenz-Landau, Germany speldszus@uni-koblenz.de
[2] TU Darmstadt, Germany geza.kulcsar@es.tu-darmstadt.de, malte.lochau@es.tu-darmstadt.de
[3] OVGU Magdeburg, Germany sandro.schulze@iti.cs.uni-magdeburg.de

- *Code smells* qualify problematic, locally restricted code structures and anomalies in-the-small, at class- or member-level (e.g., large classes).
- *Anti-patterns* qualify architectural decay in-the-large, usually involving several classes spread over the entire program (e.g., God Classes) [Br98].

Based on this taxonomy, a precise and reliable identification of actual occurrences of design flaws requires arbitrary combinations of software metrics with adjustable thresholds, as well as code smells and anti-patterns into *compound detection rules* [Mo10]. However, most existing approaches lack a comprehensive formal foundation and a uniform, yet modular representation of such design-flaw detection rules. Instead, specifically tailored detection routines are applied for every design flaw individually, and being re-evaluated from scratch for every program version anew during software evolution.

In our work, we present a comprehensive methodology for specifying and automatically detecting design flaws in object-oriented programs. The approach utilizes a unified abstract program model comprising those high-level object-oriented code entities being relevant for a concise specification of well-known design flaws. Based on this model, compound design-flaw detection rules integrate software metrics, code smells and anti-patterns, and allow for arbitrary combinations thereof. The modular nature of the rule language allows for sharing similar symptoms among multiple rules. The corresponding pattern-matching routines derived from those rules incrementally augment the underlying abstract program model with qualitative and quantitative design-related information. This technique builds the basis for efficient design-flaw detection by systematically facilitating reuse of information among multiple detection rules, as well as between subsequent detection runs on continuously evolving programs.

Please note that our tool implementation as well as all experimental data sets are available on our GitHub site (`https://github.com/GRaViTY-Tool/`).

Acknowledgements

This work has been supported by the DFG in the Priority Programme SPP 1593: Design For Future – Managed Software Evolution (LO 2198/2-1, JU 2734/2-1).

Literaturverzeichnis

[Br98] Brown, William J.; Malveau, Raphael C.; McCormick, Hays W. “Skip”; Mowbray, Thomas J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley, 1998.

[Mo10] Moha, Naouel; Guéhéneuc, Yann-Gaël; Duchien, Laurence; Le Meur, Anne-Francoise: DECOR: A Method for the Specification and Detection of Code and Design Smells. TSE, 36(1):20–36, 2010.

[Pe16] Peldszus, Sven; Kulcsár, Géza; Lochau, Malte; Schulze, Sandro: Continuous Detection of Design Flaws in Evolving Object-Oriented Programs using Incremental Multi-pattern Matching. In: ASE. 2016.