Automated Reuse of Test Cases for Highly Configurable Software Systems

Stefan Fischer[1], Gabriela Karoline Michelon[2], Rudolf Ramler[3], Lukas Linsbauer [4],
Alexander Egyed[5]

Abstract: In this work, we report about our research results [Fi20] initially published in the journal *Empirical Software Engineering, volume 25, issue 6, pp. 5295–5332, November 2020*, `https://doi.org/10.1007/s10664-020-09884-x`. We performed experiments on test reusability across configurations of highly configurable software systems. First, we used manually written tests for specific configurations of three configurable systems and investigated how changing configuration options affects these tests. Subsequently, we applied an approach developed for automated reuse, ECCO (Extraction and Composition for Clone-and-Own), to automatically generate tests for new configurations from the existing, manually written tests. The experiments showed that our generated tests had a higher or equal success rate compared to direct reuse and they generally achieved a higher code coverage. It can be concluded that automating the reuse of tests for configurable software can substantially reduce the effort for adapting existing tests and it supports a rigorous testing process.

Keywords: Variability; Configurable software; Clone-and-own; Reuse; Testing

1 Summary

Many large software systems are designed to be highly configurable with hundreds or thousands of configuration options. They introduce the flexibility to adapt the system to specific customer needs. However, some combinations of configuration options may result in undesired interactions, i.e., one option having unintended negative side effects on the behavior associated with another configuration option. Testing configurations for interactions is inherently challenging, because a large number of configuration options means that there are often myriads of possible configurations that can be derived, each showing a potentially different behavior. Combinatorial Interaction Testing (CIT) selects configurations that cover combinations of n configuration options (often referred to as n-wise testing) to reduce the amount of configurations to test to a viable number [Lo15]. Nevertheless, the actual test cases still need to be adapted to reflect the individual behavior

[1] Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria, stefan.fischer@scch.at
[2] Johannes Kepler University, Institute for Software Systems Engineering and LIT Secure and Correct Systems Lab, Linz, Austria, gabriela.michelon@jku.at
[3] Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria, rudolf.ramler@scch.at
[4] Technische Universit¨at Braunschweig, Institute of Software Engineering and Automotive Informatics, Braunschweig, Germany, l.linsbauer@tu-braunschweig.de
[5] Johannes Kepler University, Institute for Software Systems Engineering, Linz, Austria, alexander.egyed@jku.at

doi:10.18420/SE2021_08

of each selected configuration, which can result in significant extra effort for testing. Thus, Krüger et al. discussed the need for automated test refactoring for the adoption of more systematic reuse approaches [Kr18]. In the presented work, we performed experiments on test reuse across configurations of configurable software systems (RQ1) and investigate how automated reuse affects the testing effort and the resulting coverage (RQ2).

The focus of our work is on reusing existing (e.g., manually written) test cases dedicated to specific configurations of the system. These specific test cases are reused for testing other, so far untested configurations combining previously tested configurations. We applied an automated approach for reuse, ECCO (Extraction and Composition for Clone-and-Own) [Fi14], to automatically generate new variants of tests from existing ones. ECCO dissects the existing tests according to the related configuration options and combines the obtained fragments to new test cases matching the options of the new configurations under test. For our experiments, we used three different configurable systems: *StackSPL*, *ArgoUML*, and *Bugzilla* (version 3.4 and 5.1). We investigated *direct reuse* of existing tests and *automated reuse* by generating tests for new configurations, which were obtained by pairwise and three-wise combination of configuration options. The measures used for evaluation were *success rate* (i.e., the number of test cases that passed on a new configuration without adaption) and *code coverage* (as well as mutation testing for one of the systems).

Our experiments showed that a large proportion of the existing test cases could be reused on new configurations. For our first two systems, nearly all individual test variants could be directly reused, and for the two versions of Bugzilla from 70% to even 100% of tests cases could be reused. Automatically reusing tests yielded better results in success rate (avg. improvement 27.8%) and code coverage (avg. improvement 5-10.1%). These results suggest a considerable advantage for applying automated test reuse over the direct reuse, which requires additional manual effort for adapting failing test cases.

Literaturverzeichnis

[Fi14] Fischer, Stefan; Linsbauer, Lukas; Lopez-Herrejon, Roberto Erick; Egyed, Alexander: Enhancing clone-and-own with systematic reuse for developing software variants. In: 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, S. 391–400, 2014.

[Fi20] Fischer, Stefan; Michelon, Gabriela Karoline; Ramler, Rudolf; Linsbauer, Lukas; Egyed, Alexander: Automated test reuse for highly configurable software. Empirical Software Engineering, S. 1–38, 2020.

[Kr18] Krüger, Jacob; Al-Hajjaji, Mustafa; Schulze, Sandro; Saake, Gunter; Leich, Thomas: Towards automated test refactoring for software product lines. In: Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1. S. 143–148, 2018.

[Lo15] Lopez-Herrejon, Roberto E; Fischer, Stefan; Ramler, Rudolf; Egyed, Alexander: A first systematic mapping study on combinatorial interaction testing for software product lines. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, S. 1–10, 2015.