Test Case Propagation in Related Applications

John Businge
University of Antwerp, Antwerp, Belgium

Software is becoming increasingly complex and at the same time increasingly safety critical (for example in domains of robotics, autonomous systems such as cars, data-processing software, Android apps). To tackle these challenges, developers need to reuse code and collaborate effectively. Social coding platforms such as GitHub have substantially improved the situation on both the code reuse and collaboration, providing a huge bazaar of software and parts of software that can be reused; this is known as forking [1], and supported by the various facilities in those platforms like pull requests [2]. Despite the large-scale software reuse through forking, developers are not yet supported with automated techniques that can support the maintenance of these applications. Between two forked applications, GitHub only reveals number of commits a a clone is behind the other original (i.e., new commits that have not been synced). Detailed information of the updates, e.g., bug fixes and test cases and new features have to be manually identified by the developer using basic tools like in-built GitHub `diff` tool and adapted for propagation.

Illustrative example: Given a family of forked applications, suppose a critical bug is found in one of the applications, then fixed by one of the developers. Other developers reusing the clone might not know about it; even if they know, they need to manually propagate and adapt the test case as well as the code covered by the test case. Ideally, a technique could support developers by notifying them about test cases that can be adapted (e.g., checking whether the reused code can still be tested by the test case, if not, suggesting to what extent the test case can be reused and whether it needs to be adapted and how), and then automatically adapting and propagating them, perhaps interactively with the developer.

In our previous study, we conducted an exploratory study on variant-management practices in the Android app ecosystem [3]. We focused on apps that are available in the official app store, Google Play, and that host their source code on GitHub. In the carefully selected data, a total of 88 Android application families were collected. In the study, we made some considerable observations relating to artifact interdependency awareness. Relating to code sharing, the study made the following observations: 1) Applications do not often propagate code from one variant to another. Only a few mainline–fork variant pairs (38% of 127 original–fork application pairs) in the app families we studied performed code propagation. 2) Only 11% of the 127 original-fork variant pairs performed code propagation by pull requests and about 37% of the projects performed code propagation by cherry-picking of the commits. The findings of my study reveal that, after the fork date, *only a few variants in the application families share changes to existing artifacts or new artifacts*. This could be attributed to the difficulty developers face when searching for changes made in the variants that could be of interest to them. This implies that, despite the potential for it, there is limited code sharing/propagation between Android app families.

In the study I plan to carry out the following tasks:

1) *Identifying application families:* First, I will mine mainline applications (not forks) that have been forked many times and are actively maintained. Thereafter, I will mine the corresponding forks of the mainline application that are *relatively maintained but do miss some important updates from the mainline like bug-fixes and their test cases*.
2) *Test case and mutant extraction:* I will develop tools to identify a changed test case (added/modified) in one variant an application. After having identified the test cases, I will employ both dynamic and static slicing techniques to identify test-to-code-traceability—employed by Rompaey et al. [4], Qusef et al. [5].
3) *Test case and the mutants adaptation for propagation:* Here I will develop tools that will adapt the extracted test case and its mutants for propagation in the target variant. The main aim is to identify the mappings between name spaces in the source and target variant, while ensuring that the propagation carries over all the desired functionality and avoids any side-effects of regression testing. I will employ genetic programming techniques focusing on software testing as the primary driver of correctness [6].

References

[1] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 25–34.

[2] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12, 2012, pp. 1277–1286.

[3] J. Businge, M. Openja, S. Nadi, E. Bainomugisha, and T. Berger, "Clone-based variability management in the android ecosystem," in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*, 2018, pp. 625–634.

[4] B. V. Rompaey and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in *2009 13th European Conference on Software Maintenance and Reengineering*, 2009, pp. 209–218.

[5] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, "Recovering test-to-code traceability using slicing and textual analysis," *J. Syst. Softw.*, vol. 88, no. C, pp. 147–168, Feb. 2014.

[6] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke, "Automated software transplantation," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ser. ISSTA 2015, 2015, pp. 257–269.