Test Amplification on Dynamic Typed Languages A Case Study on Pharo Smalltalk

Henrique Rocha
Universiteit Antwerpen
Antwerp, Belgium
henrique.rocha@uantwerpen.be

Mehrdad Abdi
Universiteit Antwerpen
Antwerp, Belgium
mehrdad.abdi@uantwerpen.be

The importance of software testing is well-known in both academia and industry. Good testing practices can detect issues earlier on and avoid failures. Moreover, improper testing is shown to have a negative effect on software quality. In 2018, the estimated costs related to poor software quality in the US was approximately $2.84 trillion [Kra18].

Although most developers would agree that tests are important, it is not difficult to find subpar test cases on any system. Most often, manually written test cases only cover default scenarios. However, even substandard handwritten tests contain some implicit expert knowledge on the artifact being tested. Such implicit knowledge is a valuable resource that can be explored to create better test cases.

In test amplification, we use manually written tests as seeds to automatically extend existing test cases or generate new ones that improve a testing quality metric (e.g., mutation score, code coverage) [DVPY+19]. Therefore, the implicit expertise on the handwritten test is amplified to cover more situations that improve the overall quality of the test suite.

Existing test amplification tools (e.g., DSpot) focus on statically typed languages like Java [BARM15]. The static type system provides additional information that facilitates the amplification process. For example, it is possible to know that a function parameter inside a test case is an integer and, therefore, only generate integer values for it. In dynamically typed languages, such information is not available making the amplification task more challenging.

In our previous paper [ARD19], we showed that it is feasible to perform test amplification on a dynamically typed language, more specifically, Pharo Smalltalk. We created a tool called Small-Amp as a proof-of-concept. We also applied Small-Amp in a simple application to better understand the amplified tests. The lessons we learned helped guide the improvements necessary for the tool. Now we are extending the work done on the first paper to include a quantitative study on real applications and a qualitative study on real developers to assess the quality of the amplified tests. For instance, in Roassal3 (a visualization engine for Pharo), the original test cases had a mutation score of 14%, i.e., from all the generated mutants the original tests killed only 14% of them. We used Small-Amp on Roassal3 and the amplified test cases increased the mutation score to 58%. Moreover, we contacted a project manager for the Roassal3 and he confirmed that the amplified tests are indeed readable and cover more scenarios than the original ones.

For the presentation, we are going to explain the process used for test amplification in Pharo Smalltalk, a dynamically typed language, and show examples of the amplified tests on the first version of Small-Amp and now with all the improvements. We are also going to discuss the new qualitative and quantitative studies to show that we can perform test amplification on real systems.

References

[ARD19] Mehrdad Abdi, Henrique Rocha, and Serge Demeyer. Test amplification in the pharo smalltalk ecosystem. In *Proceedings of the 14th Edition of the International Workshop on Smalltalk Technologies*, IWST '19, pages 1–7, 2019.

[BARM15] Benoit Baudry, Simon Allier, Marcelino Rodriguez-Cancio, and Martin Monperrus. Dspot: Test amplification for automatic assessment of computational diversity. *CoRR*, abs/1503.05807, 2015. Available at http://arxiv.org/abs/1503.05807.

[DVPY+19] Benjamin Danglot, Oscar Vera-Perez, Zhongxing Yu, Andy Zaidman, Martin Monperrus, and Benoit Baudry. A snowballing literature study on test amplification. *Journal of Systems and Software*, 157:110398, 2019.

[Kra18] Herb Krasner. The cost of poor quality software in the us: A 2018 report. Technical report, Consortium for IT Software Quality (CISQ), 2018. Available at http://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report.