Empirical Software Engineering (2022) 27: 96
https://doi.org/10.1007/s10664-021-10094-2

Developer-centric test amplification

The interplay between automatic generation human exploration

Carolin Brandt[1] · Andy Zaidman[1]

Accepted: 2 December 2021 / Published online: 2 May 2022

Abstract
Automatically generating test cases for software has been an active research topic for many years. While current tools can generate powerful regression or crash-reproducing test cases, these are often kept separately from the maintained test suite. In this paper, we leverage the developer's familiarity with test cases amplified from existing, manually written developer tests. Starting from issues reported by developers in previous studies, we investigate what aspects are important to design a developer-centric test amplification approach, that provides test cases that are taken over by developers into their test suite. We conduct 16 semi-structured interviews with software developers supported by our prototypical designs of a developer-centric test amplification approach and a corresponding test exploration tool. We extend the test amplification tool DSpot, generating test cases that are easier to understand. Our IntelliJ plugin *TestCube* empowers developers to explore amplified test cases from their familiar environment. From our interviews, we gather 52 observations that we summarize into 23 result categories and give two key recommendations on how future tool designers can make their tools better suited for developer-centric test amplification.

Keywords Software testing · Test amplification · Test exploration · Test generation · Developer-centric design

1 Introduction

Testing is an important (Whittaker et al. 2012), but time-consuming activity in software projects (Beller et al., 2015a, 2015b, 2019). Automatic test generation aims to alleviate this effort by reducing the time developers spend on writing test cases. The software engineering community has created a plethora of powerful tools, that can automatically

Communicated by: Hadi Hemmati

Carolin Brandt
c.e.brandt@tudelft.nl

Andy Zaidman
a.e.zaidman@tudelft.nl

[1] Delft University of Technology, Delft, Netherlands

generate JUnit test cases for software projects written in Java. For example, a widely known tool is EvoSuite (Fraser and Arcuri 2011), which generates test cases from scratch using search-based algorithms. It starts from a group of randomly generated test cases and optimizes them by mutating their code and combining them with each other. This paper focuses on *test amplification*, a technique that automatically generates new test cases by adapting existing, manually written test cases (Danglot et al. 2019a). The state-of-the-art test amplification tool DSpot (Danglot et al. 2019b) mutates the setup phase of manually written test cases and generates new assertions to test previously untested scenarios. For both EvoSuite and DSpot, studies have shown that the tools are effective in generating or extending test suites to reach a high structural coverage and mutation score (Danglot et al. 2019b; Fraser and Arcuri 2011; Rojas et al. 2015; Serra et al. 2019).

Automatic test generation is, for example, used to detect regressions (Robinson et al. 2011), reproduce crashes (Derakhshanfar et al. 2020b, 2020a), uncover undertested scenarios (STAMP 2019b) and generate test data (Haq et al. 2021). For these use cases, it is often sufficient to keep the generated test cases separate from the manually written and maintained test suite (STAMP 2019b; Nassif et al. 2021). This separation is reinforced by several hard-to-solve challenges that limit the understandability of the automatically generated tests, such as their readability (Daka et al. 2015; Grano et al. 2018) or generating meaningful names (Zhang et al. 2016; Daka et al. 2017), or documentation (Roy et al. 2020; Panichella et al. 2016; Bihel and Baudry 2018).

The amplified test cases created by DSpot are closely based on manually written ones. This opens up the chance to generate test cases that are easier to understand by developers, as they are likely familiar with the original test case, which the amplified test case is based on. In this paper, we want to leverage this aspect. We take a look at generating amplified test cases that developers can take over into the manually maintained test suite as if they would have written them themselves. To describe this kind of test generation we use the term *developer-centric*, as the developer accepting the test case is central for this kind of test generation:

> Test amplification is *developer-centric* if it aims at generating test cases that are accepted by the developer and taken over into the manually maintained test suite.

Generated test cases that are accepted by developers and are part of the maintained test suite also fulfill several further typical uses for developer tests. For example, as a form of executable documentation (Hoffman and Strooper 2003; Beck 2003; Kochhar et al. 2019), or to locate the fault that causes a failing test by understanding the test in question (Panichella et al. 2016).

To provide amplified test cases that developers take over into their test suite, the interaction of the developer with the test amplification tool in which they review the proposed amplified test cases is critical. In past projects, users of DSpot reported that the tool was complex to configure and they had to wait long for the tool to finish and for them to see results (STAMP 2019b). There is little support that guides developers through the list of generated test cases so they can effectively judge whether to keep or discard a newly amplified test case. To address these issues and realize developer-centric test amplification, we embed the test amplification tool in a so-called *test exploration tool*:

> A *test exploration tool* forms the interaction layer between the developer and the test amplification tool. It lets the developer start the test amplification tool, and later explore and inspect the different amplified test cases.