**Protect Test Chapter 1 Skeleton**

Introduction

When adding features to existing software, tests will often break, necessitating their repair as part of the development process. Repairing these broken tests is a task that takes up a considerable portion of development time.<Support with citations, check waterfall paper> The typical work pattern goes something like this: Implement feature or bug fix, run test suite, examine test failures, determine if failure is due to a newly outdated test or actual app functionality has been broken, and repair either test/functionality or both.

There have been some strategies and techniques developed and implemented to automate the resolution of these problems, all of which have been ex post facto ([1]–[3]). There are some problems with the ex post facto approach. The primary problem is that the intent of the developer making changes cannot be known, and therefore, any post-facto automatic repair cannot be done without the developer’s review and approval of the proposed repair. Additionally, reviewing these proposed repairs can itself be a length and tedious task. While this semi-automated approach is an improvement over manual repair, could there be an alternative approach that is more palatable from the developer’s perspective?

With this project, we ask, and attempt to answer, whether we can take the information present in a test, and notify a developer of a potential breakage, either prior to or at the time of the code edit responsible for the breakage. This would provide the developer not only with the foreknowledge that changing some code would break a test, but also notify the developer of the breaking change as it occurs. With this knowledge, the developer can not only be certain that his/her broken test is intentional and necessary, but they can also immediately repair the test, resulting in a smoother development process.

The concept for this project was conceived following a review of Selenium [4] test breakages for web applications [5]. Selenium tests fall into the category of “record/replay” tests, whereby the perspective of the user using a web application can be affirmed through tests which are first written by recording actions on a browser, and then ensuring that those same actions can be replayed on the browser following some development and modification. After studying why these tests break, it was noticed that often, the breakages are due to single typographical changes such as changing the name of an html tag or attribute. Rather than going through an ex post facto repair process, a more streamlined approach is to notify the developer before-hand that they would be breaking a test, and allow them to repair the test at the time of breakage.

This focus of this project is therefore Selenium tests, as accidental breakage of tests is a demonstrated problem [5], and their structure and implementation are well suited to demonstrate this approach. The project is implemented as a plugin to the popular text editor Sublime Text. When a user edits HTML that will result in a Selenium test being broken, the user is notified of this fact, with an option to undo the changes leading to this breakage. <Include evaluation results, roadmap>

[1] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso, “WATER: Web Application TEst Repair,” in *Proceedings of the First International Workshop on End-to-End Test Script Engineering*, 2011, pp. 24–29.

[2] M. Hammoudi, G. Rothermel, and A. Stocco, “WATERFALL: An Incremental Approach for Repairing Record-replay Tests of Web Applications,” in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 751–762.

[3] B. Daniel, T. Gvero, and D. Marinov, “On Test Repair Using Symbolic Execution,” in *Proceedings of the 19th International Symposium on Software Testing and Analysis*, 2010, pp. 207–218.

[4] “SeleniumHQ.” [Online]. Available: http://docs.seleniumhq.org/. [Accessed: 04-Mar-2017].

[5] M. Hammoudi, G. Rothermel, and P. Tonella, “Why do Record/Replay Tests of Web Applications Break?,” in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2016, pp. 180–190.