Regression Testing of Web Applications using Record/Replay Tools

Mouna Hammoudi [1]
Department of Computer Science and Engineering
University of Nebraska - Lincoln, USA
{mouna}@cse.unl.edu

ABSTRACT

Software engineers often use record/replay tools to enable the automated testing of web applications. Tests created in this manner can then be used to regression test new versions of the web applications as they evolve. Web application tests recorded by record/replay tools, however, can be quite brittle; they can easily break as applications change. For this reason, researchers have begun to seek approaches for automatically repairing record/replay tests. This research investigates different aspects in relation to testing web applications using record/replay tools. The areas that we are interested in include taxonomizing the causes behind breakages and developing automated techniques to repair breakages, creating prevention techniques to stop the occurrence of breakages and developing automated frameworks for root cause analysis. Finally, we intend to evaluate all of these activities via controlled studies involving software engineers and real web application tests.

CCS Concepts

•**Software and its engineering** → *Software verification and validation;*

Keywords

Web Applications, Regression Testing, Record/Replay Tools

1. INTRODUCTION

Web application developers frequently employ record/replay tools to enable the automated testing of their applications. A record/replay tool for web applications captures inputs and actions (mouse clicks, keyboard entries, navigation commands, etc.) that occur as a web application is utilized. During playback, these inputs and actions are re-delivered to the browser engine. The importance of such tools is underscored by the number that exist in the research and commercial realms.

An advantage of creating web application tests via record/replay tools involves the ability to reuse them to regression test the web applications as they evolve. Unfortunately, web application tests created by record/replay tools can easily stop functioning as the applications evolve [10]. Changes as simple as repositioning page elements or altering the selections in a drop-down list can cause such tests to break. Test breakages hinder engineers' abilities to perform regression testing. For this reason, researchers [2, 12] have recently begun devising techniques for automatically repairing record/replay tests. While these techniques are often successful, there are also many occasions on which they are not.

The goal of our research is to investigate techniques for facilitating the process of testing web applications. Our research activities include taxonomizing the causes behind breakages in web application tests and developing automated test repair techniques to repair breakages within tests. Furthermore, we intend to create prevention techniques to stop the occurrence of breakages in tests via the use of enhanced IDEs that automatically repair tests as the web application evolves. Moreover, we plan to investigate the root causes behind breakages in web application tests by creating a framework that supports the extraction of the root causes behind breakages. Finally, we aim to evaluate all of these activities via controlled experiments involving real web application tests and developers.

2. BACKGROUND

2.1 Record/Replay Tools

Record/replay tools allow test engineers to capture sequences of inputs and actions applied to a web application's GUI. The recording process creates a test script that can then be replayed in an unattended mode. There are many record/replay tools for web applications available; in our work we utilize Selenium [18], one of the flagship open-source test automation tools for web applications.

2.2 Related Work

There has been some research on automated repair of *programs* (e.g., [1, 7, 13, 17]), our work focuses on tests. There have been numerous papers on test repair. Several papers have addressed the problem of repairing unit tests such as JUnit tests or tests written in similar frameworks (e.g., [3, 4, 16]). Many researchers have attempted to repair GUI tests, which track sequences of user actions applied to an interface [14, 15], [5, 6, 8, 11]. Such approaches, however, do not address directly web tests produced by record/replay tools, although they might be adapted to do so.

Zhang et al. [21] address the problem of repairing broken *workflows* in GUI applications, where a workflow is a sequence of activities to perform a given task. This approach, however, does not attempt to repair actual tests.

Other researchers have considered problems related to record/replay tests of web applications. Stocco et al. [19, 20] investigate the

[1] **Advisor:** Gregg Rothermel **email:** grother@cse.unl.edu

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983942

automated generation of page objects that confine causes of test breakages to a single class, a form of breakage prevention.

3. GOALS AND APPROACHES

The overall goal of this research is to *propose new solutions for analyzing and maintaining tests for web applications*.

3.1 Activities

To achieve this goal, we propose the following activities. We first present the activities and then elaborate on how to achieve each one.

1. Taxonomize and understand the causes behind breakages in web application tests.
2. Develop test repair techniques to automatically repair breakages in tests as web applications evolve.
3. Develop prevention techniques to automatically prevent the occurrence of breakages within tests.
4. Design a framework that supports the determination of root causes behind test breakages.
5. Evaluate the techniques and the frameworks created in Activities 2 through 4 by using controlled studies that involve real web application tests and software engineers to assess the impact of our techniques on the maintenance effort of tests.

We next elaborate on work we have done so far and work we intend to perform in the future.

3.2 Preliminary Work

In preliminary work, we have accomplished Activity 1 and we have made some progress towards Activity 2.

Taxonomy of Breakages in Web Application Tests

We created a taxonomy classifying the ways in which record/replay tests of web applications break, based on an analysis of 453 versions of eight popular web applications for which 1065 individual test breakages were recognized [10] . The resulting taxonomy is a quantitative and qualitative assessment of the causes behind test breakages. This taxonomy can help direct researchers in their attempts to repair such tests. It can also help practitioners by suggesting best practices when creating tests or modifying programs, and can help researchers with other tasks such as test robustness analysis and IDE design.

WATERFALL: An Incremental Approach for Repairing Record-Replay Tests of Web Applications

Automated test repair approaches have focused on correcting problems that occur between releases of a web application by directly analyzing differences between those releases. Often, however, intermediate versions or commits exist between releases, and these represent finer-grained sequences of changes by which new releases evolve from prior releases. We created WATERFALL, an incremental test repair approach that applies test repair techniques iteratively across a sequence of fine-grained versions of a web application [9]. The results of an empirical study on seven web applications show that our approach is substantially more effective than a coarse-grained approach (209% overall), while not imposing excessive overhead.

3.3 Remaining Work

Our work on Waterfall [9] partly addressed Activity 2 by investigating the effect of using commits and intermediate versions to drive test repair. Our repair effort was geared towards locators, which constitute the tests' components that identify web elements to be manipulated within the user interface of the web application under consideration. We chose to focus on locators specifically because they represent the majority of breakage causes within tests (73% of the breakage causes within tests) [10]. Next, we intend to extend our work to other types of breakages, by creating repair techniques that automatically repair tests through performing a differential comparison of the original web application and the evolved one. Other approaches that we may consider include the use of machine learning techniques and heuristics to automatically suggest repairs for breakages within tests.

Activity 3 aims to avoid breakages in the first place, countering the need for test repair. To achieve breakage prevention, we will develop enhanced IDEs that automatically update tests as changes are applied to the web application under consideration. For instance, appropriate test statements could be deleted, added, modified or repositioned within the test in order to meet the new logic of the web application. Such IDEs would reduce test maintenance costs and eliminate the need for repairing breakages.

Activity 4 pertains to identifying test breakages and locating (or helping engineers locate) the proximal causes of those breakages. As we classified the causes of test breakages based on actual data, it became quite clear that the time at which the proximal cause of a breakage occurs and the time at which a breakage is detected can differ widely. To repair, avoid or prevent a breakage, it is crucial to understand its distal cause and tie it back to the proximal cause of the breakage. Our previous work allowed us to distinguish the following three classes of breakages [10]. *Direct breakages* manifest themselves precisely when the breakage cause is encountered. *Propagated breakages* do not manifest themselves immediately, but do manifest themselves later on subsequent test actions. *Silent breakages* never manifest themselves explicitly. As future work, we intend to build a framework that supports the identification of the root causes behind breakages by performing appropriate analyses, such as data and control dependence analyses within web application tests.

Activity 5 consists of evaluating our techniques and approaches via controlled studies. As should be clear from our preliminary work, this activity is actually interleaved throughout the proposed work, with empirical evaluations serving to guide further steps in the research. Furthermore, we will evaluate the efficiency of our techniques by involving software engineers in order to compare our automated approaches against their manual counterparts.

3.4 Scope

In this work, we have been considering record/replay tools such as Selenium IDE. However, our work can be extended to other types of record/replay tools given that record/replay tests present the same structure and make use of the same mechanisms (locators) to identify web elements within the user interface. While we expect to retain this focus for this dissertation, in future work we intend to consider other types of testing such as programmable web testing using Selenium WebDriver or visual web testing using Sikuli.

4. EXPECTED CONTRIBUTIONS

As described in Section 3, we expect to develop new techniques for breakage prevention, avoidance and repair within web application tests. We expect to achieve breakage prevention by designing enhanced IDEs that automatically update tests as the web application evolves. Also, we intend to create a framework that supports root cause analysis. We expect our research to reduce the costs related to root cause analysis, test maintenance, repair, etc. As future work, we are planning to enlarge our research to GUIs in general, given that our work is considering web application GUIs, which are comparable to GUIs used for other types of applications.

5. REFERENCES

[1] S. Chandra, E. Torlak, S. Barman, and R. Bodik. Angelic debugging. In *Proceedings of the International Conference on Software Engineering*, pages 121–130, 2011.

[2] S. R. Choudhary, D. Zhao, H. Versee, and A. Orso. WATER: Web Application TEst Repair. In *Proceedings of the Workshop on End-to-End Test Script Engineering*, pages 24–29, 2011.

[3] B. Daniel, T. Gvero, and D. Marinov. On test repair using symbolic execution. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 207–218, 2010.

[4] B. Daniel, V. Jagannath, D. Dig, and D. Marinov. ReAssert: Suggesting repairs for broken unit tests. In *Proceedings of the International Conference on Automated Software Engineering*, pages 433–444, 2009.

[5] B. Daniel, Q. Luo, M. Mirzaaghaei, D. Dig, D. Marinov, and M. Pezzè. Automated GUI refactoring and test script repair. In *Proceedings of the Workshop on End-to-End Test Script Engineering*, pages 38–41, 2011.

[6] M. Dhatchayani, X. A. R. Arockia, P. Yogesh, and B. Zacharias. Test case generation and reusing test cases for GUI designed with HTML. *Journal of Software*, 7(10):2269–2277, 2012.

[7] D. Gopinath, M. Z. Malik, and S. Khurshid. Specification-based program repair using SAT. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 173–188, 2011.

[8] M. Grechanik, Q. Xie, and C. Fu. Maintaining and evolving GUI-directed test scripts. In *Proceedings of the International Conference on Software Engineering*, pages 408–418, 2009.

[9] M. Hammoudi, G. Rothermel, and A. Stocco. Waterfall: An incremental approach for repairing record-replay tests of web applications. In *Proceedings of the International Symposium on the Foundations of Software Engineering*, 2016.

[10] M. Hammoudi, G. Rothermel, and P. Tonella. Why do record/replay tests of web applications break? In *Proceedings of the International Conference on Software Testing*, 2016.

[11] S. Huang, M. B. Cohen, and A. M. Memon. Repairing GUI test suites using a genetic algorithm. In *Proceedings of the International Conference on Software Testing*, pages 245–254, 2010.

[12] M. Leotta, A. Stocco, F. Ricca, and P. Tonella. Using multi-locators to increase the robustness of web test cases. In *Proceedings of the International Conference on Software Testing, Verification and Validation*, pages 1–10, 2015.

[13] S. Mechtaev, J. Yi, and A. Roychoudhury. DirectFix: Looking for simple program repairs. In *Proceedings of the International Conference on Software Engineering*, pages 448–458, 2015.

[14] A. M. Memon. Automatically repairing event sequence-based GUI test suites for regression testing. *ACM Transactions on Software Engineering and Methodology*, 18(2):4:1–4:36, 2008.

[15] A. M. Memon and M. L. Soffa. Regression testing of GUIs. In *Proceedings of the International Symposium on the Foundations of Software Engineering*, 2003.

[16] M. Mirzaaghaei, F. Pastore, and M. Pezze. Supporting test suite evolution through test case adaptation. In *Proceedings of the International Conference on Software Testing, Verification and Validation*, pages 231–240, 2012.

[17] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra. Semfix: Program repair via semantic analysis. In *Proceedings of the International Conference on Software Engineering*, pages 772–781, 2013.

[18] The Selenium Project. http://seleniumhq.org/docs/03_webdriver.html/.

[19] A. Stocco, M. Leotta, F. Ricca, and P. Tonella. Why creating web page objects manually if it can be done automatically? In *Proceedings of 10th IEEE/ACM International Workshop on Automation of Software Test*, AST 2015, pages 70–74. IEEE, 2015.

[20] A. Stocco, M. Leotta, F. Ricca, and P. Tonella. Clustering-aided page object generation for web testing. In *Proceedings of 16th International Conference on Web Engineering*, ICWE 2016, pages 132–151. Springer, 2016.

[21] S. Zhang, H. Lü, and M. D. Ernst. Automatically repairing broken workflows for evolving GUI applications. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 45–55, 2013.