Software Testing: Importance of Best Practices

CS4991 Capstone Report, 2023

August Diamond
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ald8tdd@virginia.edu

ABSTRACT
A Charlottesville-based software company chose to re-organize their large testing suite to make testing more efficient. I utilized a modern software testing management platform to set up a standard format for importing old testing instructions and creating new test cases. In doing so, I examined the company's JavaScript code base to determine the most practical way to separate testing instructions based on the underlying code. I also cross-referenced incoming bug reports on Jira to determine the need for new test case creation, or re-evaluation of old test cases. My work resulted in the importation of all old test cases to the company's new test management platform earlier than expected, greater precision in bug diagnoses, and an overall reduction in the time needed to manually test the software. Future work may involve further automation and incorporating new artificial intelligence tools in the test creation process for even greater efficiency.

1. INTRODUCTION

In a world where more and more tasks are being delegated to computers, software failures are becoming increasingly disastrous. In the past decade, software has been behind issues that would have been unimaginable in the pre-internet days of computing; from self-driving car crashes to AI models mimicking human racism. Low-quality software causes no end of issues. With the current scale of the software industry, even seemingly minor failures can take substantial tolls; if poor code causes an app with a billion users to run 20% slower than it should, millions of cumulative user hours can end up wasted within a day. For these reasons, ensuring we can trust our software to perform tasks accurately, safely, and efficiently has become of great importance.

At the company where I completed my internship, clients relied upon our services to make real-time financial decisions. Any errors in the information provided by our software could potentially incur large financial losses to our users, and result in their decision to switch to a competitor service. Given these factors, it was essential to conduct comprehensive software testing to ensure our product was free of faults. At the same time, the testing process itself incurs development costs; as maintainers of a live service, the company wanted to optimize these costs to allow updates to be pushed routinely without unsustainable overhead. My work acted as a part of balancing these processes.

2. RELATED WORKS

The work of Păsăreanu, et al. (2004) has been pivotal in the adoption of automated test generation in software. They provided three techniques by which branch-comprehensive tests can be generated for software with complex input spaces or involving many preconditions. The software produced by the

company I completed by internship with is an example of software with a complex input space, and the test analysis tools which we used to check test coverage undoubtably drew from this early report.

Also relevant is the work of Barr, et al. (2015) regarding the oracle problem in software testing, which confronts the challenging task of minimizing costs while maximizing benefits of test oracles—that is, of the automated program, or when automation is inadequate, the person who verifies that the system under test is behaving as expected. A sizable portion of the tests I worked with over the course of the internship were graphical, and thus hard to automate. These tests were in line with Barr's fourth category of solutions to the oracle problem, in which the objective is to reduce, rather than replace, human effort. I primarily achieved this through what Barr would refer to as qualitative human oracle cost reduction, which involved making manual test cases easier for humans to parse and execute.

3. PROJECT DESIGN

There were five technical objectives in the internship. First, the quality assurance team wanted to move their large test suite from an older platform to a more feature-rich test management tool. The new tool supported more modular separation of tests into steps or phases, each of which could be marked as passing or failing; this allowed for more precision in determining the point of failure. It also allowed us to separate test results by system (e.g., operating system version, monitor dimensions, etc.), so that we could ensure the product functioned as expected with the different architectures our clients may be using. Second, the old manual tests were to be rewritten in a standardized format, such that distinct steps were separated where possible, instructions were updated to reflect the current structural flow of the software, and tests were generally easier to parse. Third, we were to analyze Jira bug reports, the JavaScript code base, and other related resources to determine the need for new test cases, which we would confirm with the senior QA engineers before writing and adding to the database. This became a larger focus after we finished porting the old test cases. Our fourth objective was to determine which groups of tests could easily be automated with Selenium, with the ultimate goal of automating all user interface tests. And the final objective was to assist the senior QA engineers in beginning to automate these tests with Selenium. This was a large goal that we did not expect to finish before the end of the internship period.

There were three categories of testing performed during the internship: informal testing during the revision process, formal testing for correctness, and early automated testing for proof of concept. In the first and most common form, we as members of the QA team would informally run tests ourselves while in the process of (re)writing instructions. We would note any problems, whether related to the testing instructions or outcome, and use our observations to further improve the tests. After finishing a group of tests, we would have other software engineers who were not associated with QA run through all of them and formally log their results in the test management system. This was done to test the system itself rather than to test the quality of our test instructions, though minor revisions would occasionally come from the engineers' feedback. This was also performed less than the previous form of testing due to the far higher resource cost. Finally, the senior QA staff would occasionally demonstrate the Selenium-based automated tests they had been working on to us and other employees. This form of testing was more a proof of concept, as automated testing was still in its early phases at this company and had not been formally deployed yet.

4. RESULTS

The changes to the testing instructions were well received by engineers at the company. They reported that the testing process was smoother and more efficient than it had been using the old system, reportedly taking less time to complete manual test execution. Senior QA engineers found the new test management system immediately useful for its rich features like OS-dependent results tracking, and they benefited greatly from being able to apply those features immediately to the old tests we had moved. The standardization of test formats benefited both of these groups' abilities to comprehend the nature of a test at a glance, which had been difficult with the inconsistent formats used before.

The new tests we created based on artifacts like bug reports were also well received, and those addressing high-priority functionalities were quickly reviewed by the senior QA engineers before being put into the set of production tests. A small number of these new tests were rewritten by the same QA engineers if they did not fit the project requirements, or moved to non-production branches if they addressed features that were still in development.

Relative to features in development, the automated tests were not on production by the end of the internship period, but had made substantial progress. A sizable portion of the UI-related tests had been automated, and the most significant challenge at the time of my departure was ensuring that these automated tests would work on different operating systems, monitors, etc.

5. CONCLUSION

Our work improving the testing suite at this company was beneficial to all parties involved with the software. Internally, the engineers developing the product found the new test suite easier to work with, and higher-ups enjoyed reduced operational costs associated with future testing. Clients, though not directly exposed to the testing process, benefited from a higher and more efficiently verifiable product quality standard. I found this work to be personally beneficial as well. My experience at this internship taught me a lot about the software development life cycle, the different roles in a development team, and above all, the importance and benefits of testing in the software industry.

6. FUTURE WORK

Future work on the testing suite at this company will undoubtedly involve further test automation. Given the uniqueness and customizability of this application's user interface, it may not be possible, or even desirable to move away from manual testing entirely. But there is still certainly room to improve efficiency through some level of automation. Part of this process may include researching past applications of Selenium or other automation frameworks on similarly complex UIs. Alternatively, the company may decide to keep UI-based tests manual and focus on enhanced back-end testing. Regardless, automation will surely play a larger role in the future of this company's testing process.

Another avenue for further test enhancement is generative AI. Though practical applications are relatively unexplored due to the newness of the technology, generative AI has many theoretical applications for software testing. For example, AI could be trained to generate consistently formatted manual test instructions from requirement documents, bug reports, or other relevant artifacts. It could also analyze existing codebases and test suites to identify untested or under-tested functionalities.

REFERENCES

Visser, W., Păsăreanu, C. S., & Khurshid, S. (2004). Test input generation with java PathFinder. International Symposium on Software Testing and Analysis. https://doi.org/10.1145/1007512.1007526

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering, 41(5), 507–525. https://doi.org/10.1109/tse.2014.2372785