Test-Driven Development (TDD) – Challenges and Potential Pitfalls

Nishan Adhikari, Sayeed Sajal
Department of Computer Science
Minot State University
500 University Ave W, Minot, ND, 58707
Sayeed.Sajal@minotstateu.edu

Abstract

Test-driven development(TDD), also known as test-first development, is an important component in the widely adopted agile development or extreme programming practices of today. It is a design philosophy which attempts at discovering and finding system defects as early in the development process as possible. In this sense of the word, programmers have adopted some form of test-driven development since the conception of software. However, it is only recently that a more systematic approach to TDD has been adopted widely. This work is a survey of TDD techniques developed and formalized in the past decade. From unit testing, regression testing to the more recent concept of continuous testing we evaluate the potential pitfalls of different TDD approaches, particularly as the scope and the number of contributors in a project increases.

1. Introduction

Beck's mantra of Red (write a failing test), Green(make the test pass taking any means necessary) and Refactor(clean up duplication in your code as a result of making things green) explains the smallest iteration in the TDD process. The first step of writing a test for any small sub-problem in a project serves a distinct purpose besides testing. It forces the developer to fully understand the problem at hand, and leaves them with a better sense of design, within the scope of the problem. As a result, implementations become highly decoupled, and in general, more thought out [1]. It is difficult to determine to what scope any development process is Test Driven. Particularly in large projects with many moving parts, it may be the case that not all additional functionalities are preceded by a corresponding test to that functionality, which may only be added for purely testing purposes after some portion of the problem has been attempted by the developer. It is worth noting however that in an agile process, or in fact in an iterative and incremental process of software development, it is crucial that tests be "shifted-left". Additionally, in a

continuous integration and development environment where testing and code development happen in very close proximity almost seemingly simultaneously, more and more of the design decisions on part of the developer are guided by tests.
The goal, however, isn’t essential to classify an approach with a neat label, but rather to understand some of the under-the-hood activities, and to appreciate the nuances. This paper will mainly focus on TDD presenting a discussion of the pitfalls, challenges, and scope of testing. We will also look at problems that affect testing in general, and how they affect the TDD process.

2. Developer Productivity in TDD

Among the various academic and industrial case studies done about TDD, it is generally found that TDD development takes longer. The increase in time for development is significant as shown by Bhat et al. In the two case studies performed at MSN and Windows, a test-driven approach took 15% and 35% more time respectively [10]. The same study also concluded that the resulting software developed had higher quality. This trend of increased quality at an expense of time is the same across the board. In a different study by George et. al. it was found that development took 16% more time [11]. When we try to better understand the increase in the time it takes to develop a software using TDD practices it may be good to ask ourselves whether this increase in time is a sole result of the increase in quality, or are there other factors that cause it? Is it as simple as better product more time, or are there other factors tied to the TDD process that make it slower? Much of the existing literature on developer productivity and efficiency indicate varying results. In an aforementioned study, 78% of the developers said that they found themselves to be more productive while following TDD [11]. Another study suggests that there is no significant evidence to reject the hypothesis that productivity in a test-driven approach is the same as productivity in the test last approach [12]. And yet there are other research studies which indicate that developer productivity does go down [13][14]. Since individual productivity and efficiency are qualitative in nature they can be difficult to measure and research results highly depend on the underlying methodology, the metrics used for measurement and the environment being measured. There is not one correct method of measuring productivity, and the varying results observed can be a result of different research methodologies. It is also important to consider the fact that too many developers involved in the case studies were more accustomed to testing the last approach and thinking TDD can be a bit of a learning curve and unproductivity can be looked at as an outcome of the developer’s efforts in overcome the learning curve. Having said that there is a perceived unproductivity in developers who adopt TDD, partly due to the idle time in waiting for tests to complete. However, this problem isn’t specific to TDD and affects other processes as well. The extent to which waiting for test results can be a hindrance in the development process and some of these causes for them are explained in the following section.

A. Regression Testing In TDD

As a software project gets bigger, the number of tests that have to be run to ensure build quality take on a massive scale. When any change is introduced in a section of the code, testing has to be performed on two fronts. Firstly the immediate area that has been affected by the change has to be tested. But at the same time, many of the dependent areas associated with the changed block have to be tested as well. Running these 'back' tests, also known as testing for regressions is a complication in the testing process which can be very resource intensive. Ideally, in a world devoid of resource limitations, all test suites would be run after any sort of change in the system to ensure perfect quality. However, it is infeasible to do so. A lot of research effort has been put into devising regression testing methods that can identify the most relevant tests that have to be run based on any particular change in the code [4]. The ability of a software team to successfully handle regressions can be integral to the completion of a software project. Even more so in software projects which continuously integrate new features and functionality to an existing product. A key observation to be made about regression test selection in the context TDD is perhaps that test cycles in the TDD process have to be run more often than in a test the last approach hence a successful test selection technique is crucial to handling regression in TDD. In this effect, it might be worthwhile to take a closer look at various methodologies for testing regressions.

B. Test Suite Prioritization

It is the process of reordering test suite execution in a relevant manner. Each test suite is assigned a priority value which causes high priority tests to be run earlier in the regression testing process. While the main purpose of such prioritization is to enable early detection of faults, the criteria for prioritization can vary depending on the needs of the project. Test Suites may be ordered based on code coverage, maximizing the rate of fault detection or minimizing the cost of running the test suite [5][6]. Test suite prioritization is a safe testing process that traditionally includes all sets of Test Suites relevant to the code change [4], however, the execution of the ordered test may be terminated arbitrarily once the desirable faults are detected preventing the costly execution of the entire test suite [7].

C. Regression Test Selection

In contrast to the Test Suite Prioritization, Regression Test Selection methods do not focus on running the entire test suite. A suitable subset of the test suite is selected using various heuristics and is executed completely. Regression Testing hinges on the trade-off between the cost of selecting and running a subset and the thoroughness of the test suite [8], i.e how likely are the chosen tests able to identify code changes that break the build. Accurate and robust regression test selection has more overhead than prioritization as it isn't just an ordering of the execution of the test sequences. Decisions regarding the selection of a subset of the test suite have to be made after a thorough analysis of program and system dependencies as well as the control flow [8].

D. A Shift Towards Data Analysis for Test Selection

There is not one standard of practice when it comes to regression testing. Companies often tend to take a hybrid approach making use of both prioritization and test selection methods in different stages of the development process. The Heuristics for test selection and prioritization differ widely as well. Especially in continuous integration environments for example at Google the regression testing process is very dynamic and happens in multiple phases. Separate tests are conducted before code submission and before integration [4][15]. As described by Elbaum et al in [4] Google uses a selection algorithm based on the execution and failure windows of the different test within the test suite for their tests before submission, along with a prioritization scheme which aims to order tests for efficient error detection after code submission.

The general trend for test selection and prioritization, however, has begun to change from traditional heuristic approaches to a more dynamic process relying on analysis of test data, and generating models that select and prioritize tests. A prime example of such an approach was introduced at facebook [16]. With advancements in machine learning and sufficient availability of existing test execution data, such approaches seem highly feasible and effective. At Facebook, they observed that a predictive model was capable of reducing testing costs by a factor of two while maintaining a 99.9% detection of faults [16].

3. Further Research and Shortcomings

Since this paper was a survey paper discussing some pitfalls of TDD we have only presented the solutions to the problems in general and haven't performed any quantitative analysis of the effectiveness of the suggested solution. We believe that a quantitative analysis of various regression testing methods in relation to developer productivity can prove useful. Also, the field of test selection not limited to TDD can be an avenue for work in data analysis and knowledge discovery, and there is much room for the development of predictive models for test selection. Google, in particular, has made a vast amount of test data available for researchers to apply data analysis models and techniques on the data sets.

4. Conclusion

The scale of a project affects the developers' ability to design and develop in a purely test-driven manner. While it is apparent that development of units can be very effectively handled in TDD, we cannot say for sure how well the approach scales, and whether it is feasible to maintain TDD practices in projects with enormous code-bases and a large number of pre-existing tests. Testing for regressions can become increasingly time-intensive particularly in the refactoring stages of TDD. Test selection and prioritization techniques discussed in this paper can effectively help reduce the consumption of time and resources in TDD. However, one may rightfully ask if relying on test selection strategies is a deviation from TDD practices and is simply a way to avoid the due diligence required by the process. In this regard we suggest that regression test selection at least in the refactoring stages of TDD does not affect the underlying motivation of better design through testing first and only help speed up the refactoring stage and in turn help in applying TDD in successfully completing big projects, while still maintaining the design philosophy of TDD. Predictive test selection approach seems the most feasible in this regard, given there is sufficient data to perform such a selection, as with these methods one can better measure the effectiveness of test selection.

References

[1] Beck, K. (2003). Test-driven development: by example. Addison-Wesley Professional.

[2] Rothermel, G., & Harrold, M. J. (1996). Analyzing regression test selection techniques. IEEE Transactions on software engineering, 22(8), 529-551.

[3] Graves, T. L., Harrold, M. J., Kim, J. M., Porter, A., & Rothermel, G. (2001). An empirical study of regression test selection techniques. ACM Transactions on Software Engineering and Methodology (TOSEM), 10(2), 184-208.

[4] Elbaum, S., Rothermel, G., & Penix, J. (2014, November). Techniques for improving regression testing in continuous integration development environments. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 235-245). ACM.

[5]Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2002). Test case prioritization: A family of empirical studies. IEEE transactions on software engineering, 28(2), 159-182.

[6]Bryce, R. C., & Memon, A. M. (2007, September). Test suite prioritization by interaction coverage. In Workshop on Domain-specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting (pp. 1-7). ACM.

[7] Yoo, S., & Harman, M. (2012). Regression testing minimization, selection, and prioritization: a survey. Software Testing, Verification, and Reliability, 22(2), 67-120.

[8] Biswas, S., Mall, R., Satpathy, M., & Sukumaran, S. (2011). Regression test selection techniques: A survey. Informatica, 35(3).

[9] Saff, D., & Ernst, M. D. (2003, November). Reducing wasted development time via continuous testing. In 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003. (pp. 281-292). IEEE.

[10] Bhat, T., & Nagappan, N. (2006, September). Evaluating the efficacy of test-driven development: industrial case studies. In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering (pp. 356-363). ACM.

[11] George, B., & Williams, L. (2004). A structured experiment of test-driven development. Information and Software Technology, 46(5), 337-342.

[12] Munir, H., Wnuk, K., Petersen, K., & Moayyed, M. (2014, May). An experimental evaluation of test-driven development vs. test-last development with industry professionals. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (p. 50). ACM.

[13] Mäkinen, S., & Münch, J. (2014, January). Effects of test-driven development: A comparative analysis of empirical studies. In International Conference on Software Quality (pp. 155-169). Springer, Cham.

[14] Hilton, R. (2009). Quantitatively evaluating test-driven development by applying object-oriented quality metrics to open source projects. PDF]. Available: http://www. nomachetejuggling. com/files/tdd_thesis. pdf, M. Sc thesis, Department of Computer & Information Sciences, Regis University, CO.

[15] Ziftci, C., & Reardon, J. (2017, May). Who broke the build?: Automatically identifying changes that induce test failures in continuous integration at Google scale. In Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (pp. 113-122). IEEE Press.

[16] Machalica, M., Samylkin, A., Porth, M., & Chandra, S. (2018). Predictive Test Selection. arXiv preprint arXiv:1810.05286.