Test Driven Development

Ryan Devlin

Franklin University

MIS 320

Professor Jones

12/11/16

In the traditional software development life cycle, testing would be the last phase before the code was put into production. Typically, this testing was done manually, was slow, and missed many bugs hidden in the code. These missed defects would finally make an appearance in production, causing many problems. Because these problems were found farther in the development life cycle, they were harder and more expensive to fix. This, combined with the trend towards the Agile development methodology, has led to the growth of Test Driven Development. This paper will cover what Test Driven Development is and isn’t, the benefits of using Test Driven Development, and the perceived downsides to Test Driven Development.

**What is Test Driven Development?**

Test Driven Development can be attributed to the early days of computing, when mainframe code was entered onto a punch card. Because programmers had little time with the machine, they would figure out the expected output ahead of time, and then compare the output with their findings. This would let the programmers know if they’re code was correct. The rediscovery of Test Driven Development for modern programming can be attributed to Kent Beck, who through his Extreme Programming movement, along with the Agile development movement, propagated Test Driven Development throughout the developer communities. (Barber, 2012). In Test Driven Development, the goal is to write tests before the developer writes the code the test covers. After the test is written, the developer runs the test, which should fail, given there is no code. When the developer has a failing test, he or she write the least amount of code possible to make that test pass. Once the test is passing, which may take a few cycles of fixing errors, the developer should refactor the code. Refactoring is the process of rewriting code that does not change the code’s functionality. After refactoring, the developer reruns the test to make sure everything is working as expected. By having a passing test before refactoring, the developer can make changes to the code, enhancing the quality of the code, while having faith that the functionality is unchanged. This red-green-refactor cycle, red for failing test, green for passing, is the essence of Test Driven Development, but how does that help projects?

**Benefits of Test Driven Development**

Test Driven Development is not simply a testing strategy, but a way designing and developing code. When a developer writes tests first, it requires he or she to have an understanding of the requirements of the code about to be written. By writing only enough code to make a test pass, the developer will not add unnecessary code, which adds to the overall complexity of the system, making it harder and more expensive to change or maintain. Although not all scenarios will be caught before writing code, new tests can be added to the test suite when new scenarios are discovered. This also highlights another benefit of Test Driven Development. In the beginning, tests are used to design the code being written, but once the code is implemented, the test suite is used as a regression test suite. Often, teams employing Test Driven Development will have a continuous improvement server that runs tests every night. This helps catch any defects introduced into other areas of the code base. By finding defects earlier in the software development life cycle, the cost, both technical and financial, goes down. Another benefit of Test Driven Development is the encouragement to write tests at all levels of abstraction. This means writing tests from unit level, tests that cover functionality of a single component of a system, to user acceptance level, which covers the entire workflow a user is likely to encounter when using the software. This not only helps ensure that each component is working as expected, but that all components are working together properly. Test Driven Development is not without its issues, however.

**Issues with Test Driven Development**

One issue with Test Driven Development is the amount of time and effort spent up front. This can make development feel cumbersome. (Hill, 2015). With practice and good code writing habits, the up-front time can be reduced. That introduces another flaw with Test Driven Development. Many teams not currently engaged in Test Driven Development, have little to no testing experience. Skill uplift can be expensive and slow, making it hard to sell to upper management when starting a new project. Justifying Test Driven Development after implementation can be tough as well. Because of the differences in each project, it is difficult to point to any single metric to validate the time spent up front. Only over time does the reduction in system testing and maintenance start to pay off.

Test Driven Development is not just a testing strategy for software development. It is a way of looking at the overall design of code, by writing tests before any actual code is written. This allows developers to produce higher quality code faster, with overall lower costs. However, the lack of skill in current teams makes Test Driven Development a tough sell. Lack of concrete metrics and decreased initial velocity can make Test Driven Development seem worthless from a project management perspective. These problems, although valid, can be improved over time as skills increase. The decrease of defects, along with shorter development times, can be hard to highlight, but overtime will prove to be worth the effort.

References

Barber, D. (2012, May). Why Test Driven Development. Retrieved from <http://derekbarber.ca/blog/2012/03/27/why-test-driven-development/>

Hill, S. (2015, Feb). The pros and cons of test-driven development. Retrieved from https://leantesting.com/test-driven-development/