Is Test-Driven Development Useful to Game Developers? A Case Study Exploring Applications to Data-Heavy Games

COMP130 - Software Engineering

1703170

March 20, 2018

Test-driven development is becoming increasingly popular in the software engineering industry. It is professed by practitioners of test-driven development that it can improve code quality and maintainability and that automated tests can combat regression. However, there is a noticeable lack of the practice in the games industry. This paper explores the reasons behind this, as well as the appropriateness of the practice for data-heavy games.

1 Introduction

There is much dispute in the games industry as to whether or not TDD is well suited for game development. The purpose of this paper is to discuss the merits and weaknesses of TDD in a software engineering environment and how transferable these are to game development; with particular reference to its usefulness in data-heavy games such as The Sims 4 or Magic: The Gathering.

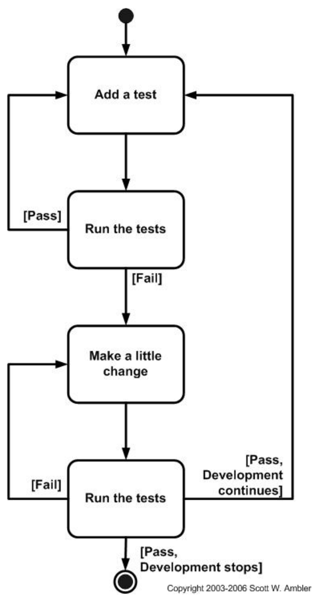
Add a test
[Pass]
Run the tests
[Fail]
Make a little change
[Fail]
Run the tests
[Pass, Development continues]
[Pass, Development stops]
Copyright 2003-2006 Scott W. Ambler

Figure 1 - The basic steps of TDD

2 What is Test-Driven Development?

Test-Driven Development (TDD) is an agile development practice [1]. It is essentially the act of writing tests for production code before writing enough production code to fulfill that test. If the tests fail then refactor until the tests pass and continue this cycle throughout the development process (see fig.1) [2].

There are a number of ways that programmers test and verify their code. Programmers can run the output to check if the application does what it's supposed to or manually check a program, using a debugger to step through code and verify it runs properly from the code base. The TDD way is to continuously create automatically running tests in parallel with production code to check that it is doing what the programmer wants.

3 The Merits of TDD

With traditional development methods, the process is as follows: design the program, implement the code, manually test the code. In this instance, manual testing is at the forefront of how a piece of software gets checked for bugs. This however does have it's drawbacks. For example, this manual testing will need to occur at the end of the implementation phase of every feature added to the program. This is fine for small programs, however the bigger a project becomes the more features are added. For every new feature, the likelyhood that another piece of code is adversely affected increases (regression defects affecting existing functionality). More testing is required to find and fix new bugs, thus lengthening development time and cost exponentially [3] (see fig.2). Studies show us that developers use 30-40 percent, on average, more effort than is initially estimated at the start of a project [4]. With more agile iterative development techniques such as TDD, the cost in development can be significantly lessened, making more frequent testing far more feasible. A study at IBM is an example of the reduction in the risk of regression defects with the use of TDD compared to other approaches [5]. A project using TDD can be far more maintainable and changeable later on nearer the end of development.

Another benefit to TDD is how it affects the psychological welfare of the programming team. Often programmers in a team taking a more traditional approach to development may go weeks without feedback on large features or even months depending on sprint lengths. When a features is finally tested it is likely to contain significant bugs or have created problems with other features. With TDD however, through the constant use of small automated unit tests, the programmer receives regular positive feedback in the form of their code passing these tests [6]. Of course this assumes that the tests will generally pass, but even when they don't they should be caught quickly, thus making the next set of tests more likely to pass than not. This can boost the morale of the programmers writing the code and keep the team as a whole more energised [7].

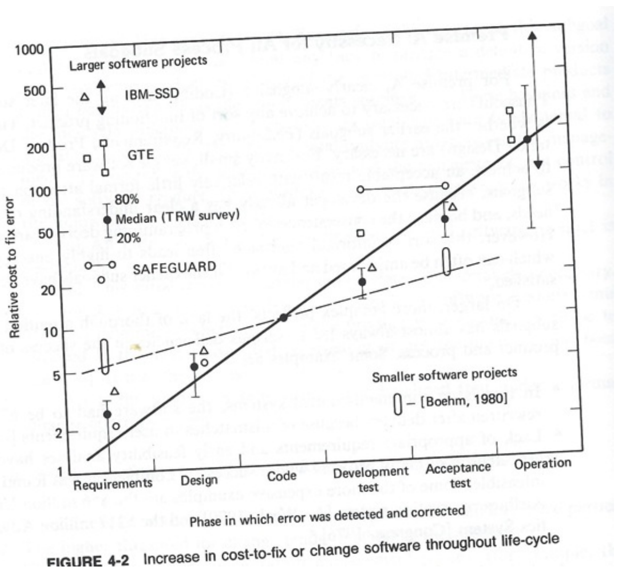

Figure 2 - Barry Boehm's original Cost of Change Curve

4 The Weaknesses of TDD

While it has been shown that TDD can save a company time and money with regards to catching bugs much earlier and more frequently [8] [7], there is debate, particularly in the games industry, as to whether or not the time saved in the manual testing phase of development is outweighed by the extra cost in time it takes to write the extra code in the form or automated unit tests. It is possible that in companies where TDD fails there are other considerations to be made as to why this is. Some programmers fail to understand what TDD is [9] and it is not unreasonable to suggest that programmers new to TDD will be much less productive when first adapting to the process.

5 TDD in Game Development

There is much debate in the games industry as to whether TDD is useful to game developers. Some consider TDD to be less helpful when developing games, suggesting that ”we cannot test fun”. This possibly misses the point that advocates of TDD make however. The usefulness of TDD to game development lie not in the testing of how a game plays but in the maintainability of the architecture and good code practice. Automated unit tests in games development can be critical to fighting hidden regression [6].

TDD may be more useful in the development of certain types of games. Data-Heavy games face a particular issue in regards to generating overwhelming code complexity. This is evidenced in the making of the Sims 4 [10]. With this game, the developers needed to build a multitasking system with certain constraints to dictate if, when and how the Sim would be able to interact with multiple objects. This is a system with less tolerance for ad-hoc implementation and so anything to help with reducing code complexity and aid continuous integration is a boon to designers. As discussed before, TDD is an ideal fit for alleviating code complexity and encourages code simplicity [11]. Automated unit tests would be especially useful given the vast number of data-driven tasks and constraints concurrent interactions require.

It is easy to see how TDD would be useful in games that continue development after release. Games like Duel Links or Magic: The Gathering are card games that continually release new content with new cards that have new interactions with already implemented cards. With TDD a programmer is more safe in the knowledge that regression is less likely and previous features are better guarded with automated tests [7]. This makes maintaining a code base and adding new features that interact with the code much less complex and far easier and quicker to do.

More corporate AAA game companies have large Quality Assurance departments, so the benefits of automated unit tests can be negated to a certain extent.

6 Conclusion

In conclusion, it seems that the practices of TDD can be useful to game developers, but how strictly the rules of TDD are followed and how useful they are depends on the type of game being made. The best practices in the game industry tend to use similar techniques to those in TDD to reap the same benefits [12] without the extra time sink and so TDD can sometimes be seen as an unnecessary annoyance. TDD is well suited for games with complex functional interactions however. Perhaps TDD will become more widely used in the industry with the advent of more 'games as a service' games with continuous development routes.

References

[1] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, "Manifesto for agile software development," 2001.

[2] K. Beck, *Test-driven development: by example.* Addison-Wesley Professional, 2003.

[3] B. W. Boehm *et al.*, *Software engineering economics.* Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[4] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Transactions on software engineering*, vol. 33, no. 1, 2007.

[5] L. Williams, E. M. Maximilien, and M. Vouk, "Test-driven development as a defect-reduction practice," in *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on.* IEEE, 2003, pp. 34–45.

[6] B. Schofield, "Embracing fun: Why extreme programming is great for game development," 2007. [Online]. Available: https://www.gamasutra.com/view/feature/130120/embracing_fun_why_extreme_.php

[7] S. Houghton, "Backwards is forward: Making better games with test-driven development," 2006. [Online]. Available: www.gdcvault.com

[8] P. Kayongo, W. Chigona, and Z. Mabhena, “Why do software developers practice test-driven development?” in *Advances in Computing and Communication Engineering (ICACCE), 2016 International Conference on.* IEEE, 2016, pp. 357–361.

[9] D. Janzen and H. Saiedian, “Does test-driven development really improve software design quality?” *Ieee Software*, vol. 25, no. 2, 2008.

[10] P. Ingebretson, “Concurrent interactions in the sims 4,” 2014. [Online]. Available: www.gdcvault.com

[11] C. Amrit and Y. Meijberg, “Effectiveness of test driven development and continuous integration–a case study,” *IT professional*, 2017.

[12] D. Fucci, H. Erdogmus, B. Turhan, M. Oivo, and N. Juristo, “A dissection of the test-driven development process: Does it really matter to test-first or to test-last?” *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 597–614, 2017.