Agile Testing Four Page Draft

Chris M. Thomas
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
thom3706@morris.umn.edu

ABSTRACT

Needs to be written!
passages or ideas worth putting into my abstract below

When it comes to software development perhaps one of the most important and time consuming processes is that of software testing. In fact, some early studies on software testing estimated that software testing could consume fifty percent or more of development costs for a product.

Currently there is much debate in the testing world about whether or not test first testing or test last testing is superior and what advantages one type of testing may offer over the other. The goal of this paper is to therefore attempt to resolve this debate as much as possible by analyzing research data in the field concerning the advantages and disadvantages of each of these testing methods.

Due to these issues new methods are starting to appear in the agile community that contain some of TDD's main tenets but have shifted enough away from TDD at some fundamental level that they are starting to receive their own names and classifications.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Delphi theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

When it comes to software development, perhaps one of the most important and time consuming processes is software testing. In fact, some early studies on software testing estimated that it could consume fifty percent or more of development costs for a product [**?**]. because of this businesses over time have become increasingly interested in attempting to optimize testing to reduce development costs.

Although there are many testing methods that currently exist two particular testing methods are currently very popular: test-last testing and test-first testing. Test-last testing, used mostly in process oriented development, is a testing method where testing is done after the software code has been written to ensure that the code works correctly. Test-first testing, used mostly in interval or sprint oriented development, is a testing method where testing is done before the software code is written to ensure that the future code meets certain requirements.

Currently, there is much debate in the testing community about whether or not test first or test last testing is superior. The goal of this paper is to attempt to give an overview of the current state of this debate by analyzing current research data concerning the advantages and disadvantages of each of these testing methods. Also because research often lags behind current implementations and the field of test first testing is currently changing due to its relatively new implementation this paper will also explore where new test first methods are going and compare them to the research given. This paper will therefore discuss the advantages and disadvantages of test first testing versus test last testing and explore new emerging test-first methodologies in an attempt to determine the applicability of each methodology.

The rest of the paper is divided into four sections which will discuss the specific ideas above in more detail. In section two of the paper we will discuss what software testing is and current software development models with their supporting testing methods: test-first and test-last testing. In section three we will provide an analysis of the data explaining the potential advantages and disadvantages of test first testing compared to test last testing. In section four we will go over new test first based testing methodologies and in section five we will state our conclusions and suggestions for further research in the field.

2. BACKGROUND

2.1 Software Testing

Software testing, simply defined, is a branch of software engineering that entails a series of practices meant to either identify potential malfunctions or demonstrate functionality in a software system [1]. Software testing can be as simple as running a program to see its results or can be as complex

UMM CSci Senior Seminar Conference, December 2013 Morris, MN.

as writing code to simulate scenarios in the real world.

When it comes to judging the effectiveness of certain testing methods it is important to note that there is no one standard quantitative measurement. Because of this many different types of measurement are used to attempt to prove that one testing methodology is better then another. Although there are many different attributes that are used for arguing the superiority of one method over another we will only focus on three attributes that are commonly found throughout research; code coverage, time, and code correctness. Code coverage refers to how many lines of code percent-wise are currently being tested. For example, if I write a test with 70% test coverage it means that 70% of the lines of code I wrote are being tested. Time simply means how long it takes to write or run the actual tests. Code correctness refers to how many errors are found within code after it is considered finished. These three attributes are popular to use because they can be quantitatively measured and are considered important within the testing community.

2.2 Waterfall Development and Testing

From as early as the 1970s a popular software development model was the waterfall model where software was developed in a series of phases. The waterfall method has been a popular development model because it is considered simple to implement correctly and is also considered to be time efficient because completing previous phases helps latter phases become more efficient. The first phase in the standard waterfall model was the requirements phase where requirements were set by the customer or design company. Next the design phase occurred where the product was designed which was then followed by the implementation phase where the product was actually created. The final phase of the development phase was then the verification phase where testing and debugging occurred wikipedia waterfall model. Due to the fact that these processes often put testing at the end of development, a certain type of testing method, the test-last method, became very popular to use with the waterfall development method.

Test last testing is currently a popular testing method and is usually the first testing method that people tend to implement. Test Last testing is the practice of writing tests after code has been written that check the functionality of their code. These tests are then used to check the written code for any potential errors which are then fixed by the developer until no errors are found by the tests. Test last testing is considered simple to implement and can allow the programmer to focus on writing tests based on areas of his code that he may be concerned about.

2.3 Agile Oriented Development and Testing

In the late 1990s some software developers started to criticize the sequence-oriented waterfall model, complaining that it was too brittle and inflexible to meet the demands of the standard customer. In response to these critiques of the waterfall model a new model for developing software emerged, the agile development model. This new development model, based on the tenets of the agile manifesto cite manifesto here!, promoted the idea that all actions of development should not occur in an ordered sequence but in a series of time-boxed iterations where in each iteration setting requirements, design, implementation and testing would occur simultaneously. The goal of each iteration is to produce a demonstrable working product to show a customer and receive feedback on the product. Some current development practices that are considered agile are Extreme Programming and Scrum.

Due to the changes in Agile programming test-last testing was pushed aside in favor of a new style known as test-first testing. In 1996 might be 2001 with the release of the agile development practice Extreme Programming, the idea of test first testing, implemented in test driven development, started to become popular for the first time [2]. Test-first testing is the practice of writing tests before code has been written and then writing code to make the tests pass. It should be noted that since tests occur before development test-first testing tends to be heavily linked to development methods and thus the most common test-first models also include development elements as well.

The most well known and most used test-first model is that of test driven development or TDD for short. In its current use in the field the phrase TDD is often used as a blanket term for any sort of development practice that uses test-first testing to drive code development. That being said it is important to note that although TDD has been turned into a vague term in this paper we will focus more on its original specific definition.

3. ANALYSIS OF TDD

3.1 TDD Stances and Data

Currently, TDD is the most popular testing method in the Agile community. Many supporters of TDD claim that it reduces overall time spent on a product, improves test coverage, and increases overall code quality. More importantly there exists data to back up some of these claims. for example, in the experiment documented by [5], it was noted that on average TDD methodologies increased code coverage by 40% compared to conventional testing methods. Another study, documented in [3] also suggested some positive traits about TDD by claiming some research showed that TDD practices could reduce effort by up to 27%. Studies in articles [3, 2, 4] acknowledged multiple studies that empirically showed at least a marginal increase in product quality when TDD was implemented. By looking at this data alone it would be relatively easy to see why TDD has such a positive image in industry and academia.

That being said, TDD also has many detractors as well who would tell you that TDD actually increases time spent on a product while not increasing overall code quality. This side also has a fair bit of research backing its opinion. For example, in [5] the same study that showed that TDD increased test coverage by 40%, it was noted that TDD took significantly more time to implement then conventional testing and did not actually increase code quality. Also, the same study that found research showing the TDD could reduce effort by up to 27% found other research that suggested that TDD actually increased the effort by two fold [3]. It is also worth noting that articles [2, 4] also have data and reports arguing that TDD takes more time then conventional testing. What is perhaps troubling about the data given here is that most of it directly contradicts the data given in the first paragraph. This suggests an interesting paradox where even though data given on TDD is statistically significant it is decidedly inconclusive.

3.2 Analysis of Conflicting Data

Even though the data surrounding TDD is inconclusive as a whole it should still be noted that information can still be extracted from it. This is especially true because many of the results were statically significant meaning that some sort of meaningful data was obtained. Because most of the data was statistically sound it may be worth considering not that the data was faulty but perhaps that the studies contained potentially key differences from one another. Thus it may be useful to us to try to understand why the data may be so conflicting as it may lead to useful insights about TDD indirectly.

One of the largest issues for summarizing the effectiveness of TDD is the field itself is very broad. It is important to note that while I specifically defined the original ideas and methodologies of the original idea of TDD, TDD itself has become an umbrella term that has been used to describe a massive array of different testing practices. This diversity may help explain why we are getting differing results because the TDD practice in one study may be extraordinary different from the TDD practice in another study. It is worth noting that in [2] a major issue with many TDD research papers was the absence of how the TDD process was implemented. This is a major issue in studying the field of TDD because it is hard to tell if people used similar or different methods of TDD to obtain their results.

Another issue for understanding the effectiveness of TDD that many TDD researchers have run into is that TDD is actually a difficult process to implement correctly. There were many testimonies throughout various articles [2, 3, 4] of industry professionals finding the TDD process to be much more difficult to implement correctly compared to traditional testing frameworks. Also, many articles on the analysis of TDD at some point brings up the topic of the complexity of TDD. This acknowledgement of complexity is important as it may suggest a result bias in TDD experiments based on the skill level of the participants as less experienced participants may be significantly less likely to implement TDD in an efficient or successful manner while more experienced participants may have more success. In one study at a college, although not significantly analyzed due to a lack of participants, it was noted that Alumni who used TDD produced overall better quality code compared to their conventional testing Alumni counterparts while current students in the college showed almost no difference in code quality between TDD and conventional test methods [5]. This potential bias would wreck havoc on TDD data conclusiveness as a whole as the two major participant types for TDD experiments are industry professionals and college students who have vast gaps in experience difference.

Overall it is very hard to draw conclusions on the effectiveness of TDD. Out of all the data out there there is really only one data point that seems to be incontrovertible which is the fact that TDD always seems to increase overall code coverage. Perhaps another conclusion that can be made is that TDD seems to produce no worse code quality then traditional waterfall testing. Also there are some implications from potential confounding factors that not all TDD methods are created equal and experience may play a large role in TDD effectiveness. Everything else on the effectiveness of TDD seems to be inconclusive.

4. EVOLUTIONS TO TDD

I feel that this is a much needed and very interesting section for exploring Agile testing but I am becoming increasingly concerned about how well this section may be able to tie into the rest of the paper. I would like to use this section as a useful resource for seeing where TDD is going/ what problems TDD developers are trying to address/ see if any of these new spin offs may solve problems that are coming up in research listed in section 3. Any input on how to tie this section into the paper or how to better reach the goals given above would be useful.

As can been seen in the above sections there are some issues in the current implementation of TDD testing. Due to these issues new methods are starting to appear in the agile community that contain some of TDD's main tenets but have shifted enough away from TDD at some fundamental level that they are starting to receive their own names and classifications. Although there are many of these spin-offs in TDD for this paper we chose to focus on three more popular spin offs, acceptance test driven development, Behavior driven development, and Agile Specification Driven Development.

4.1 acceptance test driven development

Perhaps the closest spin off to TDD, acceptance test driven development (ATDD), follows many of the ideas and expectations that regular TDD does, like writing tests first and following and using tests to define the development code. Unlike TDD though ATDD believes that in order to produce higher quality code and better customer satisfaction customers should write or define "acceptance" tests that must be passed before the product is considered finished. This part of ATDD is noticeably different from the TDD ideology because in the TDD ideology only the developer should writes the tests and the tests should be written from simplest to hardest. [2]

Although some people claim ATDD is a step in the right direction for TDD developers their are also many people who have complaints about ATDD. One of the major complaints is that ATDD is an ideal practice at best because customers will not take the time to create useful acceptance tests. [2] Yet others state that ATDD is very hard for both customers and developers to implement correctly because their is no effective common language between the two factions.(site new IEEE article here) Because of these varied complaints some of the promoters of ATDD have moved away from it to support a newer similar TDD spin off, Behavior Driven Development or BDD.

4.2 Behavior Driven Development

BDD is a new spin off of TDD that focuses on how to correctly implement the fundamental usefulness of TDD. In BDD the most important thing that it argues is that TDD is unsuccessful because it is vague and it focuses more on "testing the code" then showing the codes behavior. Thus BDD argues that solving the problem is to stop thinking about code in terms of testing but to understand code in terms of behavior. Because of this shift in ideas BDD uses different testing tools, such as JBehave for java or Cucumber for Ruby [6], compared to TDD that breaks code testing down into standard verbal sentences instead of basic testing language. The argued pros for this type of language usage is that it actually reduces much of the difficulty of tdd by

allowing us to define the purpose of our code in more natural terms thus removing much of the confusion about what is important to test in TDD and the other pro is that now it is easier to communicate goals and functionality of the code to non computer science majors (IE Managers, Customers, Sales Reps, etc.) which allows for higher customer satisfaction. this is just a blurb on BDD and what it is and what perceived advantages it contains, this paragraph will get reworked as it does not fit in the current paper well

need to do more research here

4.3 Agile Specification-Driven Development

The new testing style Agile Specification-Driven Development is a hybrid of two older testing styles TDD and DbC (design by contract).

Advantage to merging the DbC system with TDD is it gives the designer a better idea about what is important to test and other advantages should also crop up in research on DbC

need to do research here, specifically it would be useful to do some research on DbC

5. CONCLUSION

In conclusion with the current information available to us there is no obviously superior testing methods between TDD or test first ideologies and waterfall or test last ideologies. That being said it is worth noting that their are still differences between the two testing methods and that although there is not a clear overall advantage between the two languages certain traits make make one method of testing better then another method of testing in a specific scenario. For example when dealing with new and relatively inexperienced programmers the test last methodology may be more efficient then the test first methodology. Another example may be that you believe code coverage is an important necessity for your product, in this case you would probably want to use test first over test last to test your code. Therefore until research on the subject becomes more clear it would be wise to consider both testing practices as effective and potentially useful for any project you may choose to undertake.

this paragraph is a rough idea of what might be useful conclusions to draw from section three

For people who are looking to advance the field of TDD it would be potentially very interesting to see how the new TDD spin-offs such as Behavior Driven Development and Agile Specification-Driven Development compare to that of a well documented TDD process like the one suggested for Extreme Programming. Another thing that could also help advance the field is to do research comparing the effectiveness of waterfall testing and TDD testing between groups with notably different experience levels as some research tentatively points that this may be a relevant variable.

This paragraph is a rough idea on the further research topic

conclusion brain storm

-TDD increases code test coverage
-TDD perhaps takes more skill to preform
-TDD may have more effective and less effective practices
-More study is needed.
-Better study methods should be implimented

6. REFERENCES

[1] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society. *Validity: Very Good. This seems like a good article to read on testing in general. If this paper explores testing like I think it will, it may help set the tone of the entire paper and clarify important traits in testing. I think this will be a key background paper.*

[2] S. Hammond and D. Umphress. Test driven development: the state of the practice. In *Proceedings of the 50th Annual Southeast Regional Conference*, ACM-SE '12, pages 158–163, New York, NY, USA, 2012. ACM. *Validity: Good. This short article seems useful on getting the current scoop on Agile testing. I do not think that it will be a main paper unless it is very dense or relevant but I think it would serve well as a support or background paper.*

[3] T. Hellmann, A. Sharma, J. Ferreira, and F. Maurer. Agile testing: Past, present, and future – charting a systematic map of testing in agile software development. In *Agile Conference 2012*, AGILE 13, pages 55 – 63, 2012. *Validity: Good. This article I think will be a useful summary article on agile testing. This article would make a useful background article. University may not be able to access this article, in that case this article will be ignored. .*

[4] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander. A study on agility and testing processes in software organizations. In *PGood/Questionableroceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 231–240, New York, NY, USA, 2010. ACM. *Validity: Very Good. This article seems very promising as its a recent paper that has research of the advantages of agile testing compared to conventional testing. The writers seem very credible in the field. This paper is likely to be a core paper unless its actually poorly written or doesnâĂŹt mesh well with other papers.*

[5] O. A. L. Lemos, F. C. Ferrari, F. F. Silveira, and A. Garcia. Development of auxiliary functions: should you be agile? an empirical assessment of pair programming and test-first programming. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 529–539, Piscataway, NJGood/Questionable, USA, 2012. IEEE Press. *Validity: Very Good. This article seems useful as it is a modern research paper on the effectiveness of agile testing practices. That being said one of the main downsides of this article is that it splits its attention between TDD and Pair programming which is concerning.*

[6] M. Soeken, R. Wille, and R. Drechsler. Assisted behavior driven development using natural language processing. In *Proceedings of the 50th international conference on Objects, Models, Components, Patterns*, TOOLS'12, pages 269–287, Berlin, Heidelberg, 2012. Springer-Verlag. *Validity: Good/Questionable. If I use this article it will be important to analyze it for credibility. This article focuses on english language like BDD testing and the advantages of that. Can be linked*

to agile testing principles through customer developer interactions. Worried how well this paper will merge with others if it becomes a core paper.