



Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Software Engineering.

Efficient Outsourcing of Test Driven Development

Project Title (type)

Avishek Shrestha

Student Name (type)

Ning Chen, Ph.D.

Advisor's Name (type)

Advisor's signature

Date

Reviewer's Name (type)

Reviewer's signature

Date

Efficient Outsourcing of Test Driven Development

Avishek Shrestha
Dr. Ning Chen
05/15/2009

Abstract

Complex software can never be 100% bug free. The practical solution is to come up with a development process which can prevent and limit the bugs from being created, thus simplifying the testing process after the development process is complete. Test driven development has been shown to reduce the number of bugs in software. Furthermore, because test driven development is based on series of unit tests, the process can be outsourced efficiently. Outsourcing allows organizations to complete the test driven development process cost effectively, economically, and practically. Test driven development is able to overcome the various obstacles associated with outsourcing by simplifying communications, overcoming language barriers by communicating through unit tests, and correlating incremental development to differences in time zones. Furthermore, test driven development coupled with outsourcing provides several advantages. By outsourcing the development function, the developers are able to focus on development with clear indicators for progress expressed in tests completed versus tests failed. When outsourced, test driven development allows the development team to focus on satisfying the functional and nonfunctional requirements and not be distracted or adversely affected by other business needs. Outsourcing test driven development also allows developers to focus on technical aspects of software code rather than interpretation of spoken languages. Ultimately, test driven development is ideally suited for outsourcing and can be done in an effective and efficient manner.

Definitions

Architecture – The structure of a software containing system, including the software and hardware components which compromise the system and the relationships between those components.

Assumption – A statement that is believed to be true in the absence of proof.

Constraint – A restriction that is imposed on the choices available to the developer for the design of a product.

Customer – A project stakeholder who requests, pays for, selects, specifies, uses, or receives the product.

Dependency – A reliance that a project has on external factors.

Entity – An item in the business domain about which data is stored and collected.

Essential – An absolutely critical requirement.

Facilitator – A person who is responsible for planning and leading a group activity.

Feature – A set of logically related functional requirements that provides a capability to the user and enables the satisfaction of a business objective.

Functional Requirement – A statement of required operational behavior that a system will exhibit.

Lag – The difference in time between time zones of one organization to another in the outsourcing function.

Nonfunctional Requirement – A characteristic that a software system must exhibit other than operational behavior.

Outsourcing – The transfer of business function to an external service provider for cost savings and operational efficiency.

Postcondition – A condition that describes the state of a system after a use case is successfully completed.

Precondition – A condition that must be satisfied before a use case may begin.

Procedure – A step-by-step description of a course of action to be taken to perform a given activity.

Process – A sequence of activities performed for a given purpose. A process can contain one or more procedures.

Quality Attribute – A kind of nonfunctional requirement that describes a quality or property of a system.

Requirement – A statement of a customer need or objective, or of a condition or capability that a software product must possess in order to satisfy an objective.

Requirement Attribute – Descriptive information about a requirement that enriches its definition beyond the statement of intended functionality.

Scope – The portion of the ultimate product vision that the current project will address. The scope draws the boundaries between what should be part of the project.

Scope Creep – A condition in which the scope of a project continues to increase uncontrollably.

Software Requirements Specification – A collection of the functional and nonfunctional requirements for a software product.

Stakeholder – A person, group, or organization that is actively involved in a project or is affected by its outcome.

User – A customer who will interact with the system directly or indirectly.

Validation – The process of evaluating a product to determine whether it satisfies customer requirements.

Verification – The process of evaluating a product to determine whether it satisfies the specifications and conditions imposed on it at the beginning of the development phase.

Vision – A long-term strategic concept of the ultimate purpose of a software product.

Table of Contents

1.0	Introduction.....	1
1.1	Definition of the Problem.....	1
1.2	Objectives of the Study.....	3
1.3	Significance of the Problem.....	4
1.4	Review of Significant Research.....	5
1.5	Assumptions and Limitations.....	6
2.0	Choosing Test Driven Development	9
2.1	Overview of the Methodology	9
2.2	Considering Other Methodologies.....	9
2.2.1	Waterfall Model.....	10
2.2.2	Cowboy coding	12
2.2.3	Agile Methods Overview.....	14
2.2.4	Scrum.....	14
2.2.5	Agile Unified Process (AUP)	16
2.2.6	Extreme Programming (XP)	18
2.3	Test Driven Development.....	20
3.0	Outsourcing Test Driven Development.....	24
3.1	Overcoming Deficiencies in Outsourcing.....	25
3.1.1	Simple Communications.....	25
3.1.2	Overcoming Language Barriers	26
3.1.3	Working Through Time Differences.....	26
3.2	Benefits of Outsourcing Test Driven Development.....	28
3.2.1	Focus on Development	28
3.2.2	Clear Indicators of Progress	28
3.2.3	Separation of Business Needs.....	29
3.3.4	Emphasis on Technical Code	30
4.0	Conclusion.....	31
5.0	References	32

Table of Figures

Figure 1: Waterfall Model.....	10
Figure 2: Scrum Development Methodology.....	15
Figure 3: Agile Unified Process	17
Figure 4: Test Driven Development.....	21
Figure 5: Test Driven Development to Overcome Outsourcing Obstacles.....	27

1.0 Introduction

Test driven development is an ideal candidate for a development methodology that can efficiently be outsourced. First, by comparing several software development methodologies and their respective advantages and disadvantages, test driven development will be shown to be well suited for outsourcing. Test driven development also has been found to reduce the number of bugs in development. When coupled with outsourcing, test driven development is a capable software development methodology for overcoming the obstacles associated with outsourcing. Additionally, outsourcing test driven development provides several benefits for the software development process.

1.1 Definition of the Problem

Complex software can never be 100% bug free. Although software developers would like to produce perfect software, the reality is that it is impossible to produce software which is completely without bugs (Faigon, 2008) . There are several possible methods to reduce the number of bugs in release candidate software. The first, and most expected, method is to test the software after the development is complete. Software testing attempts to reveal software faults by executing the program on inputs and comparing the output to the expected output (Kamsomkeat & Rivepiboon, 2008). However, software testing has several inherent drawbacks because of the nature of testing itself.

Software testing is often categorized into several activities: test design, automation, execution, and evaluation (Offutt, 2008). Not only do these activities add to the scope of the testing process, but they also must be completed in thorough detail to be effective in eliminating bugs. Because the scope of these testing activities is so expansive, it adds to the complexity of the testing process – consequently making testing more and more important to find bugs. Furthermore, software testing can quickly become expensive and labor-intensive (Beizer, 1990). The increasing scope, complexity and importance of testing in finding bugs makes testing an expensive process. After the bugs are found, they must be conveyed to the development team to be addressed and corrected. After being corrected, they must be tested for again to ensure that they have been fixed. The entire process is expensive and labor-intensive.

Fortunately, there is another approach to reducing software bugs. By encouraging use of a development process which reduces bugs, there is less reliance on the testing process to find bugs. This consequently reduces the scope, complexity, and costs of the testing process afterwards. For this approach to be effective, it's important to find a development process which has been shown to reduce bugs in the software.

1.2 Objectives of the Study

The objective of this study is to demonstrate how test driven development is an ideal candidate for a development method that can efficiently be outsourced. First, by comparing several software development methodologies and their respective advantages and disadvantages, test driven development will be shown to be well suited for outsourcing. Test driven development also has been found to reduce the number of bugs in development. The results of case studies from teams at Microsoft and IBM which implemented test driven development showed that test driven development produced decreases between 40% and 90% relative to projects that did not use test driven development (Nagappan, Maximilien, Bhat, & Williams, 2008). Although other software development methodologies have their respective strengths, test driven development has been shown to be effective in reducing bugs in released software. When coupled with outsourcing, test driven development is a capable methodology for overcoming the obstacles associated with outsourcing. This can be done because the tasks associated with test driven development are capable of overcoming the various hurdles related to outsourcing.

It is important to recognize that outsourcing as a business function has several inherent obstacles that must be overcome. Communication problems, time zone differences, cultural differences, organization and environment differences all create obstacles with outsourcing (Rudy Hirschheim, 2006). However, these obstacles can be overcome. Test driven development, as a software development methodology, is also well

suitable for outsourcing because the processes of test driven development are ideal for overcoming the various obstacles associated with outsourcing. The methods of using test driven development to overcome the obstacles associated with outsourcing will be discussed in more detail.

Lastly, outsourcing as a business function can be beneficial to both parties involved in the outsourcing if the process is handled efficiently. By working together efficiently, both parties can greatly benefit from the advantages that the outsourcing process provides. The hiring company can focus on business decisions, while the hired company can focus on development, infrastructure, and resources that are unique and focused (Rudy Hirschheim, 2006). Consequently, the focus is managing and controlling the outsourcing process so that it is efficient, practical, and viable for both parties involved in the outsourcing.

1.3 Significance of the Problem

This problem is important to address because software can never be 100% bug free. It is important to recognize and address the various methods available to reducing bugs – either through complex and comprehensive testing methods, or by utilizing development methods like test driven development which limit bugs produced in software (Nagappan, Maximilien, Bhat, & Williams, 2008). The two approaches have their respective advantages and disadvantages – which are not the focus of this paper.

The biggest advantage of test driven development is to reduce bugs during development, thus requiring less cost and effort during the testing phase. The same resources that would have been allocated to the testing phase can be better spent elsewhere. Furthermore, fewer bugs found during testing require less time fixing the bugs after testing because after the test phase is complete, the bugs have to be removed and retested. Consequently, not only is the testing process improved, but the bug fixing process is improved as well. This shows the value of emphasizing good development techniques which help to improve other processes as well.

1.4 Review of Significant Research

While considering the various methods for reducing software bugs, it becomes clear that bug reduction could be addressed by two main methods. The first method is to test the software after development. The second method is to select a development method which demonstrates its ability to reduce bugs. There is research indicating that test driven development has been shown to reduce bugs when compared to other development methodologies. However, a few other software development methodologies need to be considered and compared. It is important to consider and evaluate commonly used software development methodologies like the waterfall model, agile unified process, and extreme programming. Existing research illuminated respective strengths and weaknesses of each software development methodology.

However, after considering other software development methodologies, it became clear that test driven development exhibited characteristics that pair well with outsourcing. There is significant and plentiful research demonstrating that outsourcing is efficient. However, more telling is the fact that many small and large businesses continue to practice outsourcing because it is efficient – subsequently reinforcing the concept of the practice’s efficiency and value. Test driven development, by design, has the potential to easily overcome many of the drawbacks associated with outsourcing. Research shows that outsourcing has several drawbacks, including communication, language, and time limitations – and test driven development is able to overcome these limitations. Furthermore, when coupled with outsourcing, test driven development demonstrates several beneficial traits.

1.5 Assumptions and Limitations

In order to follow the arguments presented in this paper, a few assumptions and limitations must be understood. First, the assumption that test driven development does not require significant more resources or a larger budget than comparable development methodologies. In order to compare test driven development to other development methodologies, the advantages in processes must be acknowledged while ignoring the costs. The specific costs, estimates, and details of each specific software development methodology are beyond the scope of this paper.

Additionally, another assumption is that both organizations are familiar with test driven development and the practice of outsourcing. This assumption is necessary to evaluate the processes and methods taken to best use test driven development to overcome the obstacles associated with outsourcing. Consequently, the costs required to familiarize the organizations with outsourcing practices and test driven development must be ignored.

In addition to being familiar with outsourcing, the assumption is that outsourcing is an efficient business function (McMahon, Distributed Development: Insights, Challenges, and Solutions, 2001). There is already much research indicating that outsourcing is efficient, while simultaneously noting that outsourcing has inherent disadvantages which must be addressed. Overall, it is important to assume that outsourcing software development is more efficient and cost effective than in house software development. In fact, if outsourcing was not cost effective and efficient, then the organization would quickly discard the practice. The continuation of outsourcing is a testament to the efficiency of the practice.

In addition to the above assumptions, there are significant limitations that need to be acknowledged. One of the largest limitations is that there is no discussion for building a cost estimation model for projects. The cost estimation model would determine the costs

of project, the estimated costs of not outsourcing the project, and the estimated costs of outsourcing the project – thus enabling management to plan for outsourcing, or decide against it. Costs are often based on an estimate of project size, either by evaluating lines of codes, functions, or complexity. This cost estimation model is beyond the scope of this paper.

2.0 Choosing Test Driven Development

2.1 Overview of the Methodology

Test driven development (TDD) is software development methodology where tests are written, executed, passed, and lead to the development of more code. First, a test is written with the intention that the current code will fail the test. Next, the code is modified so that the test will pass, and then the test is run again. Finally, when the test passes, the modified code is accepted, the test is added to the test library, and the next test is written. Test driven development is ideal for fulfilling the objectives of establishing a cost effective software development plan to reduce the number of bugs.

2.2 Considering Other Methodologies

There are a variety of software development methodologies. However, only a few popular software development methodologies will be discussed and compared to test driven development. A few of the software development methodologies for comparison are the waterfall model, cowboy coding, scrum, agile unified process, extreme programming, and finally test driven development.

2.2.1 Waterfall Model

The waterfall model is one of the most well known software development methodologies. Traditionally, software development follows a waterfall model, where the results of each phase flow sequentially into the next phase (Shelly, Cashman, & Rosenblatt, 2003). The basic phases of the waterfall model include: requirements, design, implementation, verification, and maintenance – and the adjacent phases interact with each other. See Figure 1 below.

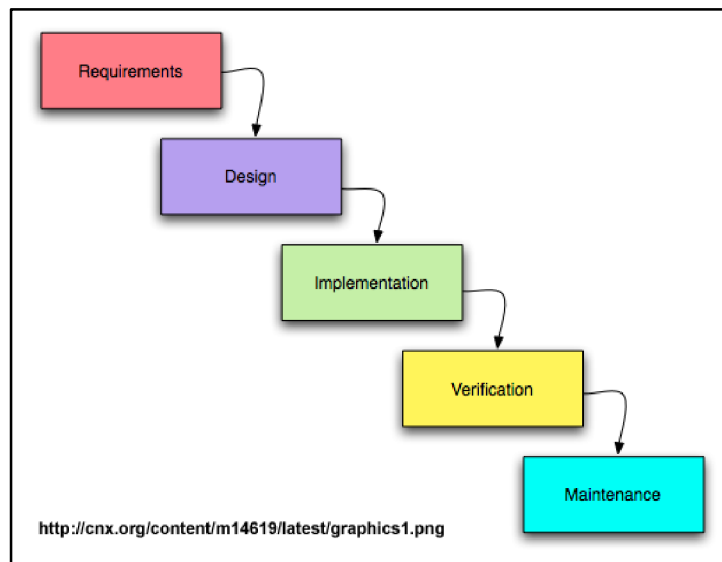


Figure 1: Waterfall Model

The waterfall model has several notable strengths and weaknesses. Having been utilized for many years, the waterfall model is well established and well known. One of the biggest advantages of the waterfall model is the emphasis and availability of documentation at the culmination and beginning of each phase (Shelly, Cashman, &

Rosenblatt, 2003). This is notable strength of the waterfall model and emphasizes the waterfall model's viability as a software development method for large scale projects or project with critical requirements. Furthermore, as one of the most well known and established methodologies, the waterfall model has been critical in providing guidance for phases in the proper order (Boehm, 1988). The waterfall model also places an emphasis on requirements analysis – thus encouraging resolution of problems in the early stages before they become bigger problems. Consequently, the waterfall model serves as a simple development model for non-changing requirements. The requirements are noted and analyzed in the early phases.

However, this emphasis on analyzing requirements also leads to the waterfall model's biggest weakness. The waterfall model is not a suitable software development methodology for software with changing requirements. Unfortunately, the real world isn't as perfect as the waterfall model specifies – requirements often change. In a practical real world environment, it's often impossible to get the previous stage 100% correct (Boehm, 1988). Additionally, the waterfall model specifies that the design is evaluated before being implemented, but that is very difficult to do (Shelly, Cashman, & Rosenblatt, 2003). Ultimately, the emphasis on carefully understanding the requirements prior to design and implementation is not realistic because requirements often change during the development phase.

Furthermore, another important weakness of the waterfall model is that the distribution of effort is heavy focused in the requirements and design phases. The waterfall model emphasizes comprehensive analysis of requirements and design prior to implementation. This leads the waterfall model to be unevenly distributed in effort. The most effort is put into the process prior to implementation. Projects with changing requirements cannot afford to place such an emphasis on requirements specifications and design up front (Wiegers, 2003).

2.2.2 Cowboy coding

Cowboy coding is often considered a rudimentary form of software development lacking in structured discipline. However, critics of cowboy coding are often misinformed and base their arguments on misconceptions. In actuality, cowboy coding is a form of software development where developers control the development without interference for guidance from management (McMahon, Bridging Agile and traditional Development Methods: A Project Management Perspective, 2004). This freedom and flexibility allows developers to control the software development using best practice methods rather than be forced to follow a rigid procedural framework. Consequently, cowboy coding is often practiced in environments of a single developer or small group of developers (Cowboy Coding, 2008).

Contrary to critics' opposition, cowboy coding has its fair share of advantages. Most obvious is the advantage that developers have the freedom to focus on development rather than adherence to organizational procedures (McMahon, Bridging Agile and traditional Development Methods: A Project Management Perspective, 2004). With this freedom, developers are allowed to place an emphasis on usable functionality rather than documentation (or other tangible artifacts). The freedom offered by cowboy coding has led to many success stories including Adobe Photoshop, Google, Apache HTTP Server, MySQL, and recently Gmail (Cowboy Coding, 2008).

However, cowboy coding has several notable disadvantages as well. Most notably, since there is no formal development process, cowboy coding is not a suitable method for large projects or projects with critical requirements (Boehm, 1988). Large projects often need requirements and specifications to be understood and analyzed prior to the development of software. Cowboy coding places on functional software rather than documentation – and that very same documentation is often emphasized at the start of large projects – which makes cowboy coding unsuitable for large projects requiring documentation. Furthermore, cowboy coding is often not suitable for projects with critical requirements because the developers are essentially responsible for identifying requirements. When a project has critical requirements, it is important for other stakeholders to be able to recognize those requirements.

2.2.3 Agile Methods Overview

Agile methods, in general, are adaptive to changing requirements (McMahon, Bridging Agile and traditional Development Methods: A Project Management Perspective, 2004). This is because the customers and other stakeholders are involved in the development. Agile methods emphasize an iterative development process characterized by: continuous engineering from requirements to test at evolving levels of abstraction; evolving the artifacts in breadth and detail based on risk assessment and priorities; and postponing completeness until later in the life cycle (Royce, 2006). These characteristics are descriptive of agile methods in general, and the details of several specific agile methods will be discussed in more depth.

2.2.4 Scrum

Scrum is an agile development method that emphasizes an approach to increase speed and flexibility in development. The word scrum comes from the rugby term – and like the rugby definition, scrum in the context of software development involves the entire team in various phases of development. There are various roles and responsibilities involved with a scrum team (Scrum (development), 2008). The “ScrumMaster” maintains the process and acts as a project manager. The “product owner” represents the interest of the customer. Finally, the “team” is the team of developers (usually 5-9 developers) where members share roles and responsibilities. See Figure 2 below.

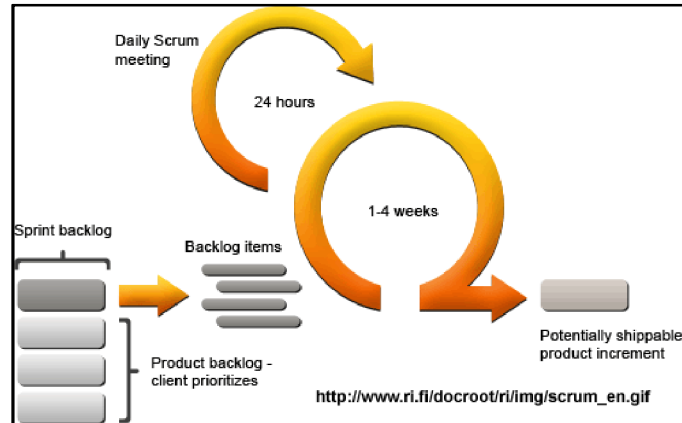


Figure 2: Scrum Development Methodology

In the scrum framework, the software development process follows several stages. The first phase is the product backlog which describes the set of requirements for the product. After the product backlog, the next phase in software development is the sprint backlog which is the set of features that will be implemented in each sprint. And then after the sprint backlog is defined, the sprint takes place. Within the sprint, the development process implements features discussed for the sprint backlog. During the sprint backlog, the backlog is not allowed to be changed until the sprint is complete. At the end of the sprint, the software product is evaluated and discussed.

Scrum has several strengths compared to other software development methodologies. Firstly, scrum is supposedly very easy to learn – requiring little more than a ScrumMaster and a self organized team. This requires very little effort to begin working within the scrum development process. Additionally, like other agile methods, scrum is

very adaptive to changing requirements because the customer is involved in the development process.

However, there are disadvantages of using scrum as well. Scrum is difficult to implement for large scale project. This is because scrum does not place an emphasis on documentation or formal maintenance, and these are often required for large projects. In contrast to the waterfall model, which emphasizes documentation and is resistant to changing requirements, scrum deemphasizes documentation is very adaptable to changing requirements. Consequently, where the waterfall model is viable for large projects, scrum is not a good solution for large projects.

2.2.5 Agile Unified Process (AUP)

The agile unified process is essentially a simplified version of the rational unified process and emphasizes rapidly putting functionally into the hands of users (Gilb, 1988). The agile unified process essentially views software change as both inevitable and desirable (Wieggers, 2003). Within the agile unified process there are six basic philosophies (Ambler, 2005):

- Your staff knows what they're doing
- Simplicity
- Agility
- Focus on high-value activities
- Tool independence
- Adaptability

The agile unified process categorizes software development into two types of releases. The first type of release is the development release which goes to a demo or staging area. After the development release is tested, revised, and finalized – it goes into the next type of release. This is the production release which goes to the production area. However, the agile unified process differs from other software development methodologies in its emphasis on mindset and attitude rather than a software development framework. The agile unified process emphasizes iterative development, a focus on architecture, and a focus on quality improvement. See Figure 3 below.

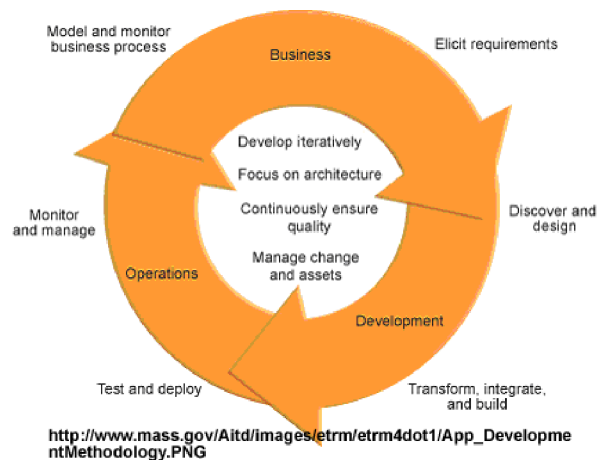


Figure 3: Agile Unified Process

The agile unified process has similar advantages to agile methods in general. It is adaptive to changing requirements because the customer is involved in the development process. Consequently, the agile unified process allows for frequent incremental and

iterative deliverables (Cockburn, 2002). It should be noted that these advantages are characteristic of agile methods in general and are not specific to the agile unified process.

However, the agile unified process also has notable disadvantages. The first and foremost disadvantage of using the agile unified process is that it is still relatively complex in comparison to other development methodologies like waterfall model, cowboy coding, or scrum. Developers may be hesitant to adopt the agile unified process because of the complexity and formality involved in the distinct processes. Additionally, the agile unified process can be relatively light on the deliverables and other artifacts that management likes to see (Gilb, 1988). The agile unified process stresses simplicity and functional software rather than documentation.

2.2.6 Extreme Programming (XP)

The last software development to be discussed is extreme programming. Unlike scrum and the agile unified process, extreme programming is a very different and distinguished approach to the agile methods. Extreme programming places a high emphasis on encouraging customers' involvement in understanding requirements – even to the point of recording requirements on the back of written index cards (Beck, *Extreme Programming Explained: Embrace Change*, 2000). Extreme programming consists of four basic activities: coding, testing, listening, and designing. Extreme programming encourages the customer to be highly involved in the development process, and encourages developers

to embrace changes in requirements at any stage. Developers are able to guide the software development process utilizing unit tests and frequent customer feedback (Beck, Extreme Programming Explained: Embrace Change, 2000). Developers assume simplicity and change by coding in small incremental, but working, changes.

Being so different from other software development methodologies, extreme programming offers a various strengths and weaknesses. Extreme programming offers the advantages of having the ability to quickly respond to changes in customer requirements at any stage at any time. The customer is actively involved in the direction of the software development. The software development is flexible and quick – which consequently reduces the cost of formal overhead. This is important to note because requirements are defined on the fly rather than attempted to be defined in advance (Beck, Extreme Programming Explained: Embrace Change, 2000). This allows software developed using extreme programming to rapidly change to changing requirements, customer demands, or market conditions.

Extreme programming, being so different from most other software development methodologies, also has unique weaknesses. Many organizations feel that extreme programming is too “light” in terms of a formal process (McMahon, Bridging Agile and traditional Development Methods: A Project Management Perspective, 2004). Because extreme programming places an emphasis on useable, functional, operational working

software, documentation is adversely deemphasized. In organizations and environments requiring documentation, extreme programming may seem too light on the documentation or formalized processes. Furthermore, because customers are so involved with the development process at all phases, extreme programming can quickly become susceptible to scope creep (Jones, 1996). Customers are encouraged to be actively involved with the development process with frequent communication with developers. This can lead to a bottleneck when customer contact is hampered or limited.

2.3 Test Driven Development

Test driven development is a software development methodology with various strengths and weaknesses which make it a viable and realistic solution for outsourcing. In comparison to other software development methodologies, test driven development is neither better nor worse for software development in general. Like other software development methodologies, test driven development has its respective strengths and weaknesses. However, in terms of feasibility when coupled with outsourcing, test driven development outshines other software development methodologies by being well suited for overcoming the obstacles that outsourcing introduces.

Within the test driven development methodology, developers create automated unit test prior to writing any code. These unit tests have must have clear pass or fail criteria so that developers can distinctly distinguish the required functionality to pass the test. After

writing the test, the developer must write the code to pass the test. If the test passes, then the developer can move on to writing the next test. If the test fails, then the code must be revised to pass the test. This basic cycle continues until the software meets the necessary functional requirements (Beck, Test-Driven Development, 2003). See Figure 4 below.

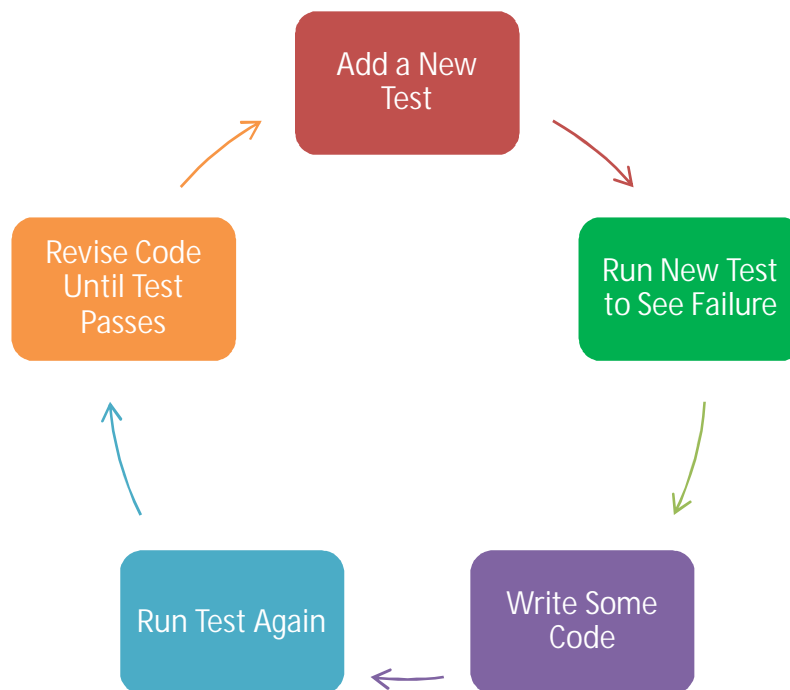


Figure 4: Test Driven Development

In practice, test driven development is slightly more complex. The unit tests are stored in a repository and serve as guidance for future software development projects. Test driven development works more efficiently as the repository of unit tests grows. Each time a new test for a new requirement or functionality is added to the repository, the developer is able to verify that the software meets all the requirements determined by previous tests except for the new requirement. Next, the developer runs a batch of

automated tests to see which fail. The developer can then write new code to pass the tests, and then rerun the batch of tests to see if they all pass. If they do not pass, then the code must be revised until they pass.

Test driven development has several strengths compared to other software development methodologies. Firstly, test driven development generally features simple code to meet functionality (Astels, 2003). The code is written to simply meet functionality and “work”, rather than written for potential (Beck, Test-Driven Development, 2003). The code must be designed for simplicity and functionality in order to pass the tests. In contrast, in other software development methodologies like the waterfall model, code is written to meet the potential and functionality of the finished product. Furthermore, by utilizing a repository of tests and revisions of source code, it is simple to revert to previous working versions of code that passed tests. The tests can also be written to imitate actual customer usage to ensure customer satisfaction with the end product.

Test driven development, being based on a collection of unit tests, is also well suited for reducing testing in the long term. A batch of automated tests help to limit bugs early in the development cycle, consequently reducing bugs (and respective costs) in the later stages (Nagappan, Maximilien, Bhat, & Williams, 2008). This encourages modularization and reused code because tests are based on smaller functional code which is tested independently. The automated tests developed through test driven development are

usually robust and can detect variations in the code (Astels, 2003). This allows test driven development to grow in complexity, scope, depth, and efficiency as more software is developed over time.

However, proponents of test driven development are aware of the criticisms as well. First, test driven development cannot be used in cases where the system must be functional prior to development. This can be the case when the software was previously developed using other software development methodologies. Additionally, the tests are part of the project overhead and must be maintained as well – although maintenance of the tests usually only requires little more than adding the tests to the repository. Lastly, because developers write both the tests and code as well, developers can be prone to including the same bugs in the tests and code. Developers who write poor tests often include the same mistakes in the code (Beck, Test-Driven Development, 2003). However, several of these weaknesses can be overcome when test driven development is outsourced.

3.0 Outsourcing Test Driven Development

Test driven development, as a software development methodology, is well suited to outsourcing. Test driven development emphasizes the usages of automated testing tools (Beck, Test-Driven Development, 2003). This automation and utilization of automated test tools makes it easy for organizations to outsource the phases of test driven development. Outsourcing, as a business function, has shown to be efficient and effective, but with an inherent set of disadvantages (McMahon, Distributed Development: Insights, Challenges, and Solutions, 2001). Test driven development is able to address several of those disadvantages. In practical terms, outsourcing must be efficient to be used.

Outsourcing test driven development can be done with little overhead. Because test driven development focuses on development through the use of test cases and the corresponding code must function in order to pass the tests, the outsourcing organization can reduce code review efforts by briefly reviewing the code and focusing on the written tests. Reviewing tests, which are usually not many lines in code, can be far more efficient than reviewing thousands of line of code. The code itself can be reviewed with less scrutiny. The code is known to be functional because it passes the tests (Beck, Test-Driven Development, 2003). Consequently, careful scrutiny of the tests can allow casual review of code. By reviewing the test cases and ensuring that the functional requirements are met, then the code itself can be briefly reviewed. If the code passes the test, then the code is sufficient (Nagappan, Maximilien, Bhat, & Williams, 2008).

3.1 *Overcoming Deficiencies in Outsourcing*

3.1.1 Simple Communications

It has been mentioned several times that test driven development is able to overcome the various obstacles associated with outsourcing. One of the primary obstacles with outsourcing is the communication – whether it's language barriers, time zone differences, or differences in cultural environments and attitude (Kannan, 2009). Test driven development, with its focus on simple unit tests, can simplify communications. Rather than discuss the nuances of complex software code, the communication can be simplified by focusing on the tests rather than the code. Tests are written to have clear pass or fail criteria and if the code passes the tests, then the code must be sufficient (Rintoul, 2009). The usual methods for overcoming communication barriers should be employed, including frequent communication, proper email etiquette and grammar, and milestone conference calls. Test driven development allows the focus to be on simple tests with distinct pass or fail criteria, rather than complex code. Furthermore, because the test driven development process is simple in concept and both parties involved in outsourcing are assumed to be familiar with the outsourcing process, communications regarding the test driven development process should be simple as well.

3.1.2 Overcoming Language Barriers

However, regardless of how simple the unit tests used for development are, there are always language barriers. Most offshore outsourcing is to countries like India, where language barriers affect communication (Business Process Outsourcing in India, 2009). Test driven development can overcome these language barriers because development is based on a collection of pass or fail test criteria rather than interpretation of documented functional requirements or software requirements specifications. Consequently, simple technical unit tests drive development and progress, rather than interpretation of documentation (Rintoul, 2009). This limits the effects that language barriers (and the ability to interpret and understand the language) have on the development.

3.1.3 Working Through Time Differences

Another problem that outsourcing introduces is the time difference between the organizations. Test driven development is ideal software development to overcome this hurdle of outsourcing because development is done iteratively in incremental bursts. New tests are added to the repository of tests, and in each increment of development, the new tests are the only tests expected to fail. Utilizing source configuration management to control revisions and revert to previously working versions of code allows test driven development to be completed in incremental stages (Astels, 2003). Consequently, any lag in time difference only affects the incremental development rather than the entire development process as a whole. When both organizations are available to communicate,

only the failed tests need to be discussed without having to necessarily consider all the passed tests. This makes communication efficient, and continues the software development progress. During lag times, new tests or development can be completed. This demonstrates the efficiency of driven development in overcoming the time difference obstacle associated with outsourcing.

Several obstacles associated with outsourcing can be overcome with test driven development. See Figure 5 below.

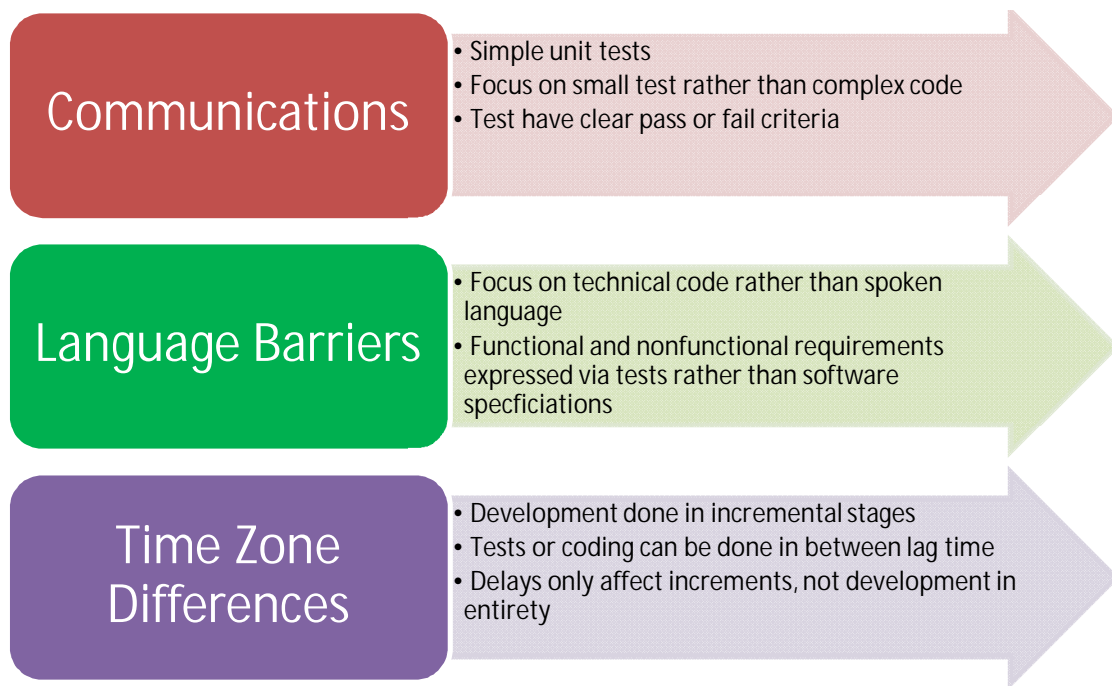


Figure 5: Test Driven Development to Overcome Outsourcing Obstacles

3.2 ***Benefits of Outsourcing Test Driven Development***

3.2.1 Focus on Development

Test driven development is complementary to the outsourcing process and introduces several benefits to software development. One of the benefits of outsourcing test driven development is that the developers are able to focus on development rather than the other aspects of the business which may serve as distractions (Rintoul, 2009). Outsourcing test driven development allows the developers to focus entirely on development – whether that development is the system in its entirety or partially. Because the developers are able to focus entirely on development rather than other business needs, outsourcing test driven development allows the developers to safely ignore if the development is part of a wider process (Rintoul, 2009). This makes outsourcing test driven development efficient because developers are unaware and unburdened about the scope of the overall development project.

3.2.2 Clear Indicators of Progress

Furthermore, the progress of development is clearly marked by tests which pass or fail. Once all the functional and nonfunctional requirements have been outlined in test cases, it is easy to see which tests fail and which pass, thus marking progression to completion of development. This is more helpful for analyzing development than having a percent complete model from a specification driven software development methodology like the waterfall model. Rather than attempt to specify the software in its entirety prior to

beginning development, and then marking progress with percent complete, test driven development takes a completely different approach by allowing iterative and incremental progress marked by clear pass or fail criteria. An iteration of development can simply be analyzed by the number of tests which pass versus the number of tests which fail.

3.2.3 Separation of Business Needs

By outsourcing test driven development, the business needs are kept separated from the software development. Simply put, if the tests pass, then the software development is sufficient. Outsourcing test driven development allows for the separation of the need to develop functional software from other influences of the business needs (Rintoul, 2009). Software developers are not subjected to the negative influences of the other business divisions, like marketing or sales for example. Instead, the development is focused on passing tests which have explicit criteria for passing or failing (Astels, 2003). Once the repository of tests completely satisfies the functional and nonfunctional requirements of the software, and the code is able to pass the tests, then the software development is complete. Outsourcing test driven development allows this to happen without the negative influence from other business needs by allowing developers to focus entirely on development.

3.3.4 Emphasis on Technical Code

Lastly, outsourcing test driven development emphasizes technical tests rather than understanding any specific language (usually English). Developers are not required to understand any “business jargon” from software requirements specifications, or the context of business needs. In other software development methodologies, like the waterfall model, the requirements are delivered through documentation like the software requirements specification. However, test driven development makes it simple to understand the functional and nonfunctional requirements by utilizing technical tests. Consequently, outsourcing test driven development eliminates the need for developers to be familiar with the language itself. The developers must simply understand the technical aspects of the tests and code, rather than the language (usually English). Outsourcing test driven development allows for the focus to be on software code rather than any specific language.

4.0 Conclusion

When compared to other software development methodologies, test driven development has been shown to reduce the number of bugs in software. This is a testament to the efficiency of test driven development in reducing bugs. Furthermore, test driven development is able to be outsourced efficiently because it is based on a series of unit tests collected into a repository. Outsourcing test driven development allows organizations to complete the software development process effectively, economically, and practically. Test driven development is able to overcome the obstacles related to outsourcing by simplifying communications, overcoming language barriers, and working through time zone differences. Outsourcing test driven development also has the benefits of allowing developers to focus on development, assess development progress easily, focus on technical code rather than interpret spoken language, and focus on the satisfying functional and nonfunctional requirements rather than be distracted by other business needs and influences. Overall, test driven development is an ideal software development methodology to be outsourced efficiently.

5.0 References

Ambler, S. (2005). *The Agile Unified Process (AUP) Home Page*. Retrieved 2008, from Amysoft Home Page: <http://www.ambysoft.com/unifiedprocess/agileUP.html>

Astels, D. (2003). *Test-Driven Development*. Upper Saddle River, NJ: Prentice Hall .

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley.

Beck, K. (2003). *Test-Driven Development*. Boston, MA: Addison-Wesley.

Beizer, B. (1990). *Software Testing Techniques*. New York, NY: Van Nostrand Reinhold, Inc.

Boehm, B. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer*.

Business Process Outsourcing in India. (2009). Retrieved 2009, from Wikipedia: http://en.wikipedia.org/wiki/Business_process_outsourcing_in_India

Cockburn, A. (2002). *Agile Software Development*. Boston, MA: Addison-Wesley.

Cowboy Coding. (2008). Retrieved 2008, from Wikipedia: http://en.wikipedia.org/wiki/Cowboy_coding

Faigon, A. (2008). *Testing for Zero Bugs*. Retrieved 2008, from <http://www.yendor.com/testing/>

- Gilb, T. (1988). *Principles of Software Engineering Management*. Harlow, England: Addison-Wesley.
- Jones, C. (1996). Strategies for Managing Requirements Creep. *IEEE Computer*, 92-94.
- Kansomkeat, S., & Rivepiboon, W. (2008). An Analysis Technique to Increase Testability of Object-Oriented Components. *Software Testing, Verification, and Reliability*, 193-219.
- Kannan, N. (2009). *Agile Outsourcing: Managing Communication Effectiveness*. Retrieved 2009, from Sourcingmag: <http://www.sourcingmag.com/content/c070604a.asp>
- McMahon, P. (2004). Bridging Agile and traditional Development Methods: A Project Management Perspective. *Systems & Software Technology Conference*.
- McMahon, P. (2001). Distributed Development: Insights, Challenges, and Solutions. *CrossTalk*.
- Nagappan, N., Maximilien, M., Bhat, T., & Williams, L. (2008). Realizing Quality Improvements Through Test Driven Development: Results and Experiences of Four Industrial Teams. *Empir Software Eng*, 289-302.
- Offutt, J. (2008). Editorial: Software Testing is an Elephant. *Software Testing, Verification, and Reliability*, 190-192.

Rintoul, G. (2009). *Offshore Outsourcing and the Value of Test Driven Development*. Retrieved 2009, from <http://www.slideshare.net/grintoul/offshore-outsourcing-and-the-value-of-testdriven-development#stats-bottom>

Royce, W. (2006). Coping with the New Paradigms. In D. Reifer, *Software Management* (pp. 55-72). Hoboken, NJ: John Wiley & Sons.

Rudy Hirschheim, A. H. (2006). *Information systems outsourcing: enduring themes, new perspectives, and global challenges*. Berlin: Springer.

Scrum (development). (2008). Retrieved 2009, from Wikipedia:
[http://en.wikipedia.org/wiki/Scrum_\(development\)](http://en.wikipedia.org/wiki/Scrum_(development))

Shelly, G., Cashman, T., & Rosenblatt, H. (2003). *Systems Analysis and Design*. Boston: Thomson Learning.

Wieggers, K. (2003). *Software Requirements*. Redmond, Washington: Microsoft Press.