Test-Driven Development (TDD) and Behavior-Driven Development (BDD): Improving Software Quality and Reducing Bugs

Swamy Prasadarao Velaga

Department of Information Technology

Published In IJIRMPS (E-ISSN: 2349-7300), Volume 2, Issue 1, (January-February 2014)

License: Creative Commons Attribution-ShareAlike 4.0 International License

Abstract

This paper aims at establishing how TDD interacts with BDD with special emphasis on the improvement of software quality and reduction of bugs. Starting with the general description of the SW testing methodologies, it explores the basic concepts and TDD's working procedures, including its historical development. Furthermore, the paper also provides understanding on BDD looking at its definition and features, its main practices and tools with poignant reference to TDD. Central to understanding of these frameworks is a critical evaluation of the various effects of TDD on level of software quality. Test-Driven Development (TDD) is a software development practice in which a developer first writes an automated test case that defines a desired improvement or new function, then produces the minimum amount of code needed to implement and pass the test, and finally refactors the code to ensure that it conforms to current design standards [1]. The advantages of TDD include improved software quality, reduced bugs, and better design of the software; the practice has also been shown to create a faster development cycle. Behavior-Driven Development (BDD) extends TDD by using natural language descriptions of the behavior of the software components under test. TDD and BDD are independent practices, and do not require specialized tools. However, to be successful with either TDD or BDD, developers must possess certain skills and knowledge. Citing real life experiences as well as academic research, the paper analyzes TDD's effectiveness in improving code quality, maintainability and shortening the cycle. Similarly, it investigates BDD's involvement in enhancing software quality, contributing to the understanding of its impact on the conformity with user requirements, acceptance and acceptance criteria, and on the rate of discovered bugs. Test-Driven Development (TDD) is a software development technique in which a programmer writes a short code that defines the desired improvement or new feature and a test case that fails until the programmer writes just enough code to make the test pass, and rearrices the code to optimal form [1]. The benefits of TDD are better quality of software and lower number of bugs, better architectural design of the software, and it was explained that TDD increases the speed of development [2]. BDD is the extension of TDD where the description is based on natural language describing characteristics of the components under test. TDD and BDD are two unrelated techniques of testing and there is no specific tool that is needed for it. Nonetheless, asserting effective TDD or BDD requires certain talents and information in the developers. This article discusses TDD and BDD, after that, claiming some of the important points that will assist the developers to get the greatest outcome for the practices. Last of all, the other considerations that the article makes regarding TDD and BDD are the approaches to testing and the use of proper testing tools.

Keywords: DevOps, Continuous Integration (CI), Continuous Deployment (CD), Infrastructure as Code (IaC) Automation, Cross-functional Teams, Site Reliability Engineering (SRE), Software Delivery, Agile Methodology, Lean Methodology, Executive Leadership, Data-driven Feedback

1. Introduction

Test-driver development, also known as TDD is a method of software development that is centered on the repetition of a very micro computer programming cycle. Primers create a test for a function/feature, write the least amount of code to pass the test in question, and then refactor the new code to best practices. TDD stems from, and is used in, the processes of Extreme Programming (XP) developed by Kent Beck and is popular because of it's the step-by-step approach that encourages simple design and instills confidence in the code as it is being developed through testing. The essence of TDD lies in its mantra: The phrase "Red, Green, Refactor," is the names of the three stages of the test first [3]. Using a test first approach means writing a failing test, making it pass and then refactoring the code. Controlling the software quality while keeping the rhythm and the flexibility is, nevertheless, always a problematic issue to manage in the software development process. The most important part is as software systems evolve and become utilized in all sectors of our economy for financial, health care and other necessary sectors, the profession requires sound and highly reliable dependable software. This requires the use of the right testing techniques that not only expose faults but also prevent the creation of these faults. Test-Driven Development (TDD) and Behavior-Driven Development (BDD) are the two approaches that were introduced with the view to improving software quality and minimizing bugs [3].

As for the second, Behaviour-Driven Development (BDD) is to be considered as the further evolution of Test Driven Development which includes the participation of technical and non-technical stakeholders into the development and testing process. Best known by Dan North in 2003 as BDD stands for Behavior Driven Development, it employs a vocabulary deduced from the system's actions, focusing on stakeholders' cooperation and relating software development with business goals [3,4]. Acceptance-driven development pays specific attention to the application's behavior from the client's perspective and typically uses frameworks such as Cucumber or SpecFlow that enable writing of tests using business language constructs such as Given-When-Then.

Hence, both TDD and BDD aim to solve typical issues, which are faced by developers, customers, and end-users, including uncoordinated requirements, detection of defects only after product launch, and various problems of maintaining sizable codebases [3,5]. These methodologies involve testing right from the initial phase and subsequently in each of the phases of development which lends a hand in identifying the defects at a period earlier than if one was to follow the traditional testing method. Additionally, they advise better design decisions and non-obfuscating, less contentious code, which are essential to software's long-term architectural integrity. Many papers have investigated the effect of TDD on software quality. Based on the surveys, it is possible to conclude that TDD makes the code more effective, increases coverage, and results in high efficiency. But it is not without challenges with the United states being in conflict with international laws on several aspects [6]. It is hailed as a possibility to have short feedback cycles, thoroughly tested code, and to improve design over time, but at first, their productivity may suffer because of TDD's alleged slow pace of development and because TDD requires a change of mindset. In addition, the quality of the tests is critical; if the tests are written improperly, then it may give a false impression of the software's readiness and overlook some defects.

BDD works hand in hand with the technical aspects of TDD while considering the behavior and interaction of the software with other users; this aspect helps in covering some of the gaps stated above because it focuses on behavior and ensures that all stakeholders engage in defining the expected behavior and testing it against the actual behavior of the product. This line with the business goals and users' expectation is a major strength of BDD making it very useful in ensuring that the software is built to meet actual needs of the users. However, BDD also involves a cultural change within the organization, which makes communication and collaboration the gene of software development [7]. Even though TDD and BDD are two different techniques, they can supplement each other and provide benefits of both techniques. Combining TDD's strong testing elements and BDD's emphasis on behavior and collaboration may sum up to effective delivery of high-quality, defect-free programs with great resemblance to what the users and the business requires. In the context of this paper, it will be essential to introduce TDD and BDD, understand how they help improve software quality, determine the differences between them, examine the main issues connected to their implementation, along with the development tendencies in this area [8]. Through the review of literature and case studies, this paper seeks to review how and why TDD and BDD can be used in enhancing quality and avoiding bugs in the creation of dependable and efficient software programs.

2. Research Problem

The main research problem in this study is to determine the best way to adopt TDD and BDD so as to promote the improvement of software quality by reducing the rate of bugs in software projects. This requires the understanding of the theory behind TDD and BDD, considering their practices and procedures; the review of available data and reports based on the observation of TDD and BDD effects on particular aspects of software quality. Some of the important issues to address pertain to code maintainability, the appearance of defects and how close the product is with the use requirements. Therefore, the objective of the study is to evaluate both TDD and BDD for the development of software to establish the advantage, limitation, and conditions suited for each technique. These dynamics should be better understood so that strategies to overcome them can be proposed and to increase the usage of these methodologies [8,9]. The study also seeks to engage future directions and recommendations for future research in relation to methodologies in software testing especially at the juncture of TDD and BDD to enhance the production of high quality software solutions that addresses user's needs and expectations. Therefore, in this vast analysis, the study aims to provide an appropriate source of direction to the practitioners, researchers, and other stakeholders in the software development field. TDD has often been found to result in relatively thorough testing. One of TDD's benefits is that programs produced using it often contain fewer bugs. Although many developers feel the resulting decrease in debugging time and bug count, little hard empirical data exists in the research literature [10. Many studies have reported various software development problems, and a number of those problems can be solved by using TDD [10]. The implementation of a solution is the main issue that researchers and developers face today. The industry requires a software development methodology that reduces bugs, increases correct implementation, and creates high-quality software products. A TDD approach has the potential to meet these industry requirements, since it is focused on testing.

3. Literature Review

A. Practices of Test-Driven Development (TDD)

Test-Driven Development (TDD) is a software development process that has proven to have three primary benefits. First, it significantly reduces the time spent on debugging. Second, it leads to an

improvement in the overall design quality of the software. Third, it acts as comprehensive documentation for all system features. When TDD is effectively implemented, the only time debugging is necessary is when a failing test does not provide sufficient information to identify the problem. In such cases, it indicates that TDD was not carried out in a "test-first manner", and this becomes the consequence of not adhering to the correct TDD approach. TDD results in a comprehensive suite of tests that exercise all system features, and it also ensures that passing code is written to satisfy each new test. Consequently, TDD effectively serves as requirements analysis, implementation, and verification[11].

The idea at the core of Test-Driven Development (TDD) is very simple: before you write some code, first write a test that proves the code is needed. Although simple in theory, this practice requires both a change in mindset and a commitment to writing tests before writing code. The following points summarize the key principles and practices of Test-Driven Development: Write a failing test before you write any code that is being tested. This is the first step in TDD. Take a small step and write the simplest code that will cause the test to pass [11]. Refactor the code to remove any duplication or other "code smells". Refactoring is the process of cleaning up your code without modifying its behavior. Refactoring is important to keep the code from deteriorating over time.

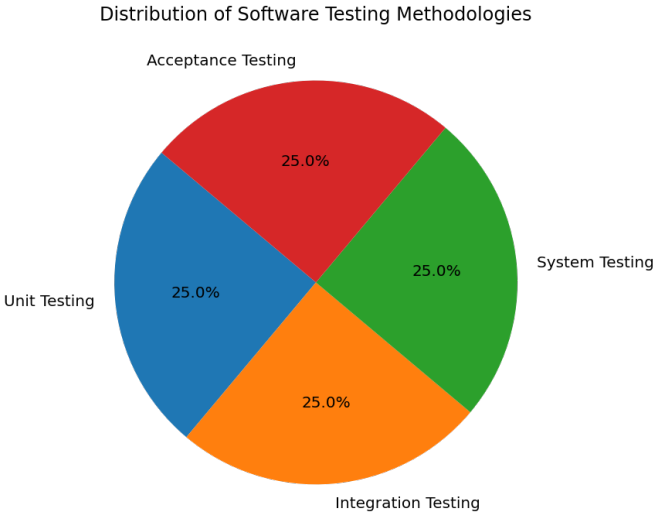

Fig. 1: Distribution of Software Testing Methodologies

B. Behavior-Driven Development (BDD)

In the software development life cycle (SDLC), understanding the requirements clearly is a major step. BDD (Behavior-Driven Development) advocates writing the actual behavior of the system from the perspective of its stakeholders. It encourages the collaboration between developers, testers, and other non-technical members of the team, such as business analysts, support, and domain experts. In BDD, the system behavior is described in natural language using a given-when-then format and is called a spec. A spec is an executable specification that provides living documentation, where documentation is always up-to-date and can be easily maintained because it is close to the code and is part of the code base. BDD helps in writing and delivering testable requirements in a test-driven development flow, thanks to behaviors defined before the implementation starts. The major strength of BDD is that it follows the principles of TDD and extends the benefits of TDD across the organization. The first person who formulated BDD and created a framework to support it was Dan North [12]. His goal was to solve the problem of communication, conversation, and misunderstanding that was still happening in TDD,

despite tests being considered as a good form of documentation. What the team was testing with TDD and what the developers were building could still be different. In BDD, the focus is on the behavior of the system. It is therefore a development process that is highly aligned with the Agile concept of working software because it is all about getting software to actually do something demonstrable and useful. The major differences between TDD and BDD are that the unit of progress in TDD is tested while in BDD it is behavior, and that TDD is implicit about what to test while BDD is explicit about what should be tested. The major strength of BDD is that it strongly aligns technical issues while TDD aligns production code with test code [12,13].

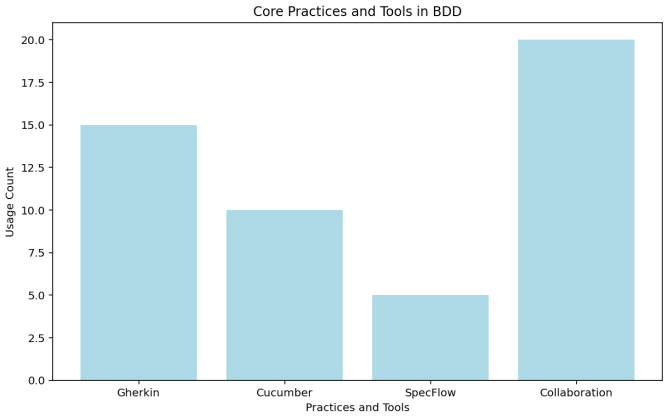

Fig. 2: Core Practices and Tools in BDD

C. Impact of TDD on Software Quality

DevOps is not just about developers and the IT operations, Test-driven development (TDD) and Behavior-driven development (BDD) are key development techniques of agile software development processes. They break development work into small increments with continuous testing throughout the development process. The tests are developed before the code, and it is required that the automated test must fail, which means there is no production code that exists to make that test pass, before writing just enough code to fulfill the test requirements. This tight feedback loop not only improves code quality, but also serves as design and documentation of the code. Various studies have taken different approaches and are largely in agreement that TDD does result in good quality code that has a lower number of bugs. There might be a few studies that hint that quality might not improve during the adoption phase of TDD, but by and large everyone agrees that when TDD is followed stringently, it does result in good quality code with fewer bugs [13].

Additionally, the quality of any software system declines when no automated test suite is available. This implies that the act of testing keeps the code healthy. When the TDD way of writing tests is followed, where tests are written at various levels of granularity ensuring complete test coverage, the resulting test suite serves as a safety net that allows all team members to freely change and refactor code knowing that the system behavior has not changed [14]. The confidence level in the code is high when a comprehensive suite of tests is executed automatically every time new code is checked-in, and this confidence leads to better quality software with fewer bugs. Therefore, automated testing is a key attribute of agile software development and TDD is the primary technique that ensures testing is an integral part of the development process.

D. BDD's Role in Enhancing Software Quality

Test-driven development (TDD) and Behavior-driven development (BDD) have become a crucial part of modern software development as it ensures higher software quality. This process not only reduces the number of bugs but also helps in reducing rework which in turn saves time. The simple fact is that the earlier you find a bug the easier and cheaper it is to fix. Therefore, catching bugs at the initial coding stage is paramount. In traditional testing, developers write tests that check the program logic for different modules along with the expected outcomes. On the contrary, in BDD, the developers write tests that check the expected behavior of the system by executing the scenarios throughout the development cycle. Such checks are implemented with the help of tools like Cucumber or JBehave that support the BDD approach.Qualitative analysis and empirical evidence suggest that BDD is a better testing method in terms of test coverage since it has a clear advantage of serving both documentation (executable specifications) and development of software. With BDD, the emphasis shifts from writing test cases to defining expected behavior in the form of scenarios from which test cases are derived automatically [14]. Although BDD is not a silver bullet, it makes TDD more systematic and enhances software quality by making the coding process closer to real user requirements and the overall process more objective.

As a simple example, writing a failing test, updating the software to make the test pass and then refactoring the code is at the heart of TDD. BDD extends TDD and is considered to be the next step because it allocates the QA team and the Business Analyst a more active role in the process and focuses on the behavior of the system being developed. The BDD approach makes the testing process more systematic as it begins by examining the software design and development process. The Three Amigos (developer, BA, and tester) come together to discuss the feature that needs to be developed and to define the feature in terms of scenarios. These scenarios are then tested by writing executable specifications[14].

E. Comparative Analysis of TDD and BDD

TDD is done in the unit testing level. BDD is done in the acceptance testing level which helps in collaborating with business. TDD expresses developer test whereas BDD expresses tester test. The unit tests written in TDD are by developers and the test scripts are called user stories in BDD, and they are written by testers. The outputs of TDD are the executable unit tests whereas the outputs of BDD are the documents like living documentation, reports, and application can also be an output in terms of automation. TDD is considered as the process of engineering whereas BDD is considered as the process of design [14,15]. Many developers believe that TDD is only the testing approach of the developers who can have the code developed. So the design of functionality given by BDD can be tested. TDD is smaller in scope whereas BDD is larger in scope. Both TDD and BDD have the same implementation part and approach also looks similar but the major difference would be the focus on the implementation.

During test-driven development (TDD), developers write tests first and then write the minimum amount of code necessary to make the tests pass. Although TDD provides many benefits, it lacks the ability to verify and validate the design of a system, application or feature, focusing instead on testing code. Designing tests that also document application's behaviors and features, while ensuring the overall quality of the test suite, is where behavior-driven development (BDD) steps in. BDD has gained popularity as an agile software development methodology enhancement alongside TDD [16,17].

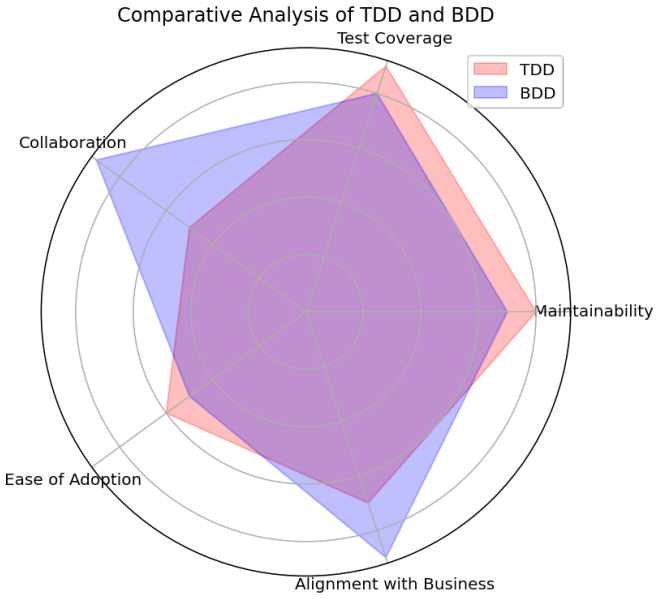

Fig. 3: Comparative Analysis of TDD and BDD

4. Contributions

My contribution in this study is to provide the transition from traditional software development to TDD/BDD for all software projects. The impact has been significant. The quality of the software I deliver has improved to the point where my customers are now considerably more satisfied than they were previously. The enhanced software quality not only increases customer satisfaction, but it also assists me in decreasing the time and expense associated with providing support. TDD requires that we should write test codes first before writing the production code. The feedback prompts me to look into the TDD activities and come up with TDD test cycles that can help both hardware and software engineers to use TDD effectively in the embedded system development, especially when the hardware is not ready. Of particular interest is improving the quality and accuracy of behavior verification as well as design and development when using BDD. Future research should more closely integrate natural language processing (NLP) with BDD processes, harnessing the power of existing and novel NLP techniques to automatically or semi-automatically accomplish tasks associated with better specifying, interpreting, executing, and developing BDD scripts. Furthermore, we envision extending TDD and BDD to incorporate data-driven development, allowing the flow of test-driven and behavior-driven principles to be used when creating, evolving, and executing data scripts in addition to behavior scripts.

No matter how comprehensive the automated test suite, developers will always witness deteriorated test and code quality over time when deciding to drop the development of tests. This situation raises the idea of integrating self-testing test suites within software systems to guarantee continuous execution and maintenance. Using in-vivo test generation allows for optional development testing and will automatically re-enable any test that is disabled or gets broken if developers try to prune the test suite. In-vivo test generation provides developers with two major benefits. First, it allows for integrating generated unit tests within production code, test harnesses, or test drivers. This ensures continuous execution of the unit tests but also enables developers to execute the generated tests in different environments. Second, in-vivo test generation allows for recorded scripts to test against different versions of the software.

5. Significance and Benefits

The TDD and BDD methods address issues of great importance and make substantial contributions. The methods are very worthy of consideration, and it is likely that you will want to incorporate some variant of them into the WS practices that you define.Test-driven development (TDD) and behavior-driven development (BDD) are very effective development practices used by professional developers all over the world. TDD is a simple practice to manage and write the code, whereas the BDD approach is more complex but yields terminal goals alignment with engineering in a bookkeeping describing the behavior of the system which needs to be developed [18]. Test-driven development and behavior-driven development close the gap between the stakeholders that are part of the software development process, including customers, and the development team. TDD and BDD bring many software development processes and product benefits regardless of some existing challenges. Programs secured by TDD are less likely to contain bugs, have excellent design quality, are easier to maintain, and are more resilient to changing requirements [19]. Programs enhanced by BDD are more probable to work as per the expectations of the stakeholders and have excellent alignment of goals.

6. Conclusion

The main aim of this paper was to perform a thorough analysis of microservices architecture with the emphasis placed on the use of the architectural model to design extensible and manageable application programs. test-driven development (TDD) and behavior-driven development (BDD) are worthwhile approaches to developing varied software projects. The value of TDD and BDD goes far beyond the number of unit tests, acceptance tests, and the test automation suite that is generated as an outcome of the testing practices. The essence of TDD and BDD is to stimulate the interaction between the stakeholders and the development team to improve mutual understanding. It is this misunderstanding that yields better software quality and reduces bugs in the end. BDD is more potent than TDD because of the broader scope of behavior description in BDD as compared to simply coding functionality of the developed system in TDD. However, both practices can be utilized together in synergy to gain the maximum benefits. By integrating TDD and BDD, developers and stakeholders can ensure that the software meets both technical and business requirements. This collaboration between different parties in the development process ultimately leads to a more robust and user-friendly software product. Additionally, the use of TDD and BDD encourages a more structured and systematic approach to software development, which ultimately leads to more reliable and maintainable code. Ultimately, TDD and BDD are essential methodologies in modern software development, providing a framework for successful collaboration, code quality, and ultimately, user satisfaction.

References

[1] G. Lee, Test-Driven iOS Development. Addison-Wesley, 2012.
[2] J. Bender and J. Mcwherter, Professional test-driven development with C#: developing real world applications with TDD. Indianapolis, In: Wiley Pub, 2011.
[3] J. Newkirk and A. A. Vorontsov, Test-driven development in Microsoft .NET. Redmond, Wash.: Microsoft Press, 2004.
[4] GärtnerM., ATDD by example : A practical guide to acceptance test-driven development. Upper Saddle River, Nj: Addison-Wesley, 2013.
[5] K. Pugh, Lean-agile acceptance test-driven development : Better software through collaboration. Boston, Mass. London: Addison-Wesley, 2011.

[6] A. Shalloway, S. Bain, K. Pugh, and A. Kolsky, Essential Skills for the Agile Developer. Addison-Wesley Professional, 2011.
[7] C. Johansen, Test-Driven JavaScript Development. Addison-Wesley Professional, 2010.
[8] D. Astels, Test-driven development : A practical guide. Upper Saddle River, N.J. ; London: Prentice Hall Ptr, 2003.
[9] S. R. Palmer and J. M. Felsing, A Practical Guide to Feature-driven Development. Prentice Hall, 2002.
[10] J. W. Grenning, Test Driven Development for Embedded C. Pragmatic Bookshelf, 2011.
[11] E. White, Making Embedded Systems. O'Reilly Media, Inc., 2011.
[12] S. Metz, Practical Object-oriented Design in Ruby. Pearson Education, 2013.
[13] G. Concas, E. Damiani, M. Scotto, G. Succi, and Springerlink. Online Service, Agile Processes in Software Engineering and Extreme Programming : 8th International Conference, XP 2007, Como, Italy, June 18-22, 2007, Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.
[14] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010.
[15] C. Møller, Advances in enterprise information systems II : Proceedings of the 5th International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2011), Aalborg, Denmark, October 16-18, 2011. Boca Raton (Fla.): Crc Press, 2012.
[16] M. M. Cruz-Cunha and J. Varajao, Enterprise information systems design, implementation and management : Organizational applications. Hershey, Pa: Business Science Reference IGI, 2010.
[17] A. Gunasekaran and T. Shea, Organizational Advancements through Enterprise Information Systems: Emerging Applications and Developments. IGI Global, 2009.
[18] A. Gunasekaran, Modeling and analysis of enterprise information systems. Hershey, Pa. IGI Publ, 2007.
[19] M. Sacks, Pro Website Development and Operations Streamlining DevOps for Large-Scale Websites. Berkeley, Ca Imprint: Apress, 2012.