



International Journal of Engineering, Business and Enterprise Applications (IJEBA)

www.iasir.net

A Study of the Software Development Using Agile

Divya Prakash Shrivastava
Computer and Information Science
Higher Colleges of Technology
Abudhabi, UAE

Abstract: Software development methodologies are constantly evolving due to changing technologies and new demands from users. While agile methods are in use in industry, little research has been undertaken into what is meant by agility and how a supposed agile method can be evaluated with regard to its veracity to belong to this category of software development methodological approaches. Here, one study conducted on some well-known agile methods. This information is shown to be useful, for when constructing a methodology from method fragments (method engineering) and when comparing agile methods.

Index Terms: Behavior Driven Development, Test Driven Development, Software Engineering, Agile Method.

I. INTRODUCTION

To develop software, companies have tried many different techniques. In the past, programming an application was thought of as a solving a problem, and the method for solving the problem was thought to only be as good as the software that was created. The object-oriented paradigm plays a prominent role in the development of many modern software systems. The different structure and behavior of object oriented software helps in solving or mitigating several problems of procedural software [1].



Fig. 1: Waterfall Development Model

Agile Development is a term used to define a modular approach to development. These incremental approaches have been used since the late 50's. The Agile philosophy focuses on communication with the client and clean, iterative development. Agile development also descended into development strategies known as Extreme Programming. Extreme Programming and Agile Development use iterations for development iterated through a full development cycle, from conception to production. Before the iterative development strategy that Agile Development utilizes, development was normally called a waterfall model (Fig. 1). Agile methodology has become the main stream of software development due to its ability to generate higher customer satisfaction. It is fast and flexible methodology that recommends to incorporating last minute changes and requirements provided by the customer at any stage of software development phase [2] [3].

II. AGILE METHODOLOGIES

Agile development provides opportunities to assess the direction throughout the development lifecycle. This is achieved through regular cadences of work, known as Sprints or iterations, at the end of which teams must present a potentially shippable product increment. By focusing on the repetition of abbreviated work cycles as well as the functional product they yield, agile methodology is described as “iterative” and “incremental.”

Agile development is supported by a bundle of concrete practices suggested by the agile methods, covering areas like requirements, design, modeling, coding, testing, project management, process, quality, etc. Some notable agile practices include:

- Acceptance test-driven development (ATDD)
- Agile Modeling
- Backlogs (Product and Sprint)
- Behavior-driven development (BDD)

- Feature Driven Development (FDD)
- Continuous integration (CI)
- Domain-driven design (DDD)
- Dynamic Systems Development Method (DSDM)
- Iterative and incremental development (IID)
- Pair programming
- Autism spectrum disorder(ASD)
- Refactoring
- Scrum meetings
- Test-driven development (TDD)
- Agile testing
- Use case
- User story
- Story-driven modeling
- Velocity tracking

Agile methods have been extensively used for development of software products and some of them use certain characteristics of software, such as object technologies.

A. Test Driven Development

Test Driven Development (TDD) is the core part of the Agile code development approach derived from eXtreme Programming (XP) and the principles of the Agile manifesto.

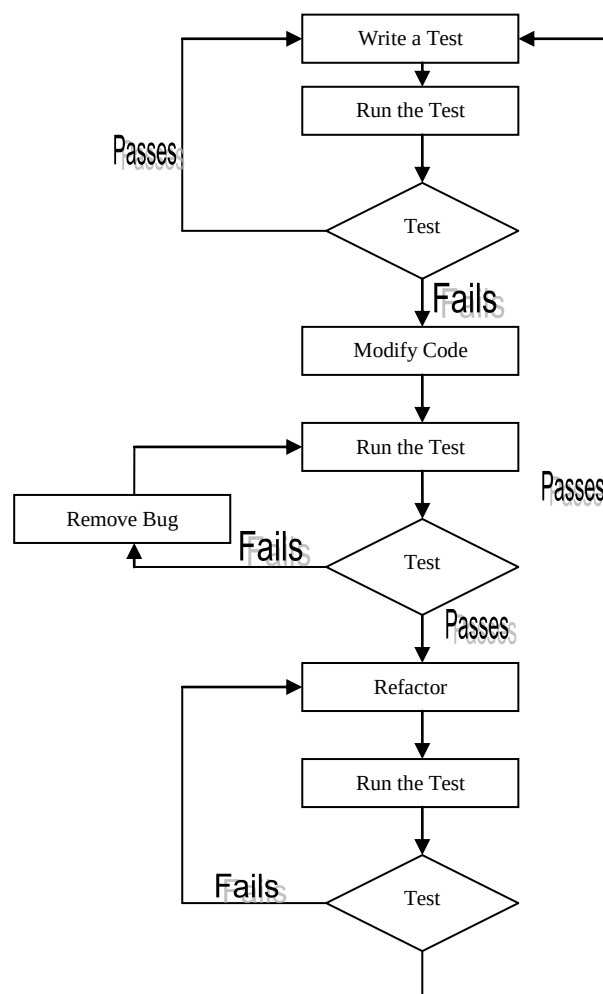


Fig. 2: Test Driven Development

The TDD is not a testing technique, rather a development and design technique in which tests are written prior to the production code. The tests are added its gradually during its implementation and when the test is passed, the

code is refactored accordingly to improve the efficacy of internal structure of the code. The incremental cycle is repeated until all functionality is implemented to final [4].

This is it. This is the simplest way of explaining TDD in my opinion. Now, the main focus of TDD will be on testing the low-level functionality and units (for single classes and methods) that can lead to more flexible code and easier refactoring. In a simple language we can say, we write these tests to check if the code we wrote works fine.

B. Agile Specification Driven Development

This is a technique that combines both- TDD and Design by Contract (DbC). DbC –this allows a language to declare invariants for the class and pre and post condition for a method. It was given by Bertrand and Meyer. Microsoft has introduced their version of DbC into .NET framework, called code contracts. Meyer has also introduced an approach called Quality-first Model to software development. Another developer described Quality approach as:

- Write code as soon as possible because then supporting tools immediately do syntax, type and consistency checking.
- Current set of functionality should be working before moving to next. Abnormal cases must be dealt with. Interwine analysis, design and implementation
- Always have a working system. Get cosmetic and style right.

The DbC approach is at core of Quality-first model. Similarities in DbC and TDD: In both the approaches one unit of functionality must be finished before moving to next. Differences: in TDD, it asks the developer to focus on the most common case, whereas DbC expects the developer to focus on abnormal cases first. Acceptance Test Driven Development. Now, let us consider another approach which is Acceptance Test Driven Development (ATDD) that adds 'A' before TDD which stands for Acceptance. Now, why was this even needed? Wasn't the TDD good enough and better? I would say no and the reason was that TDD was more of telling to make sure the code works fine but it did not say that if the code that is written was even required at first place. In ATDD the acceptance criteria are defined in early in application development process and then those criteria can be used to guide the subsequent development work. ATDD helps developers to transform requirements into test cases and allows verifying the functionality of a system. A requirement is satisfied if all its associated tests or acceptance criteria are satisfied. In ATDD acceptance tests can be automated. TDD and ATDD are adopted widely by the industry because they improve software quality and productivity [5, 6]. ATDD is a collaborative exercise that involves product owners, business analysts, testers, and developers. ATDD helps to ensure that all project members understand precisely what needs to be done and implemented. However, many developers find themselves confused while using TDD and ATDD in their projects, "programmers wanted to know where to start, what to test and what not to test, how much to test in one go, what to call their tests, and how to understand why a test fails" [6]. Some of the problems of TDD and ATDD are that they are focused on verifying the state of the system rather than the desired behaviour of the system, and that test code is highly coupled with the actual systems' implementation [8, 9]. In addition, in these approaches unstructured and unbounded natural language is used to describe test cases which are hard to understand [6].

C. Behavior Driven Development

To develop software, companies have tried many different techniques. In the past, programming an application was thought of as a solving a problem, and the method for solving the problem was thought to only be as good as the software that was created. As higher level languages evolved, software development became more popular and software became increasingly mainstream. Therefore the method that a developers used to create software came into question. Behavior Driven Development (BDD) was born out of that evolution. BDD has a strong focus on collaboration during software development, much more than it's predecessor Test Driven Development (TDD). BDD's focus does not end with developer collaboration, but instead attempts to bridge the gap between the developer and client, where features of an application are defined by how a user would interact with them rather than a feature being defined in technical terms. BDD bridges the gap between the developers and the stakeholders in software by providing a clear concise methodology to develop software. The basic work flow for software developed using BDD techniques requires the developer to communicate with the shareholders, and it requires developers to create a well defined picture of the application. Then, through the development process, shareholders communicate with the developer to shape each individual feature to the specific goals of the shareholders. This approach creates a dialogue between the developer and shareholder to create an end application that meets the specifications much more closely than applications developed in any other way.

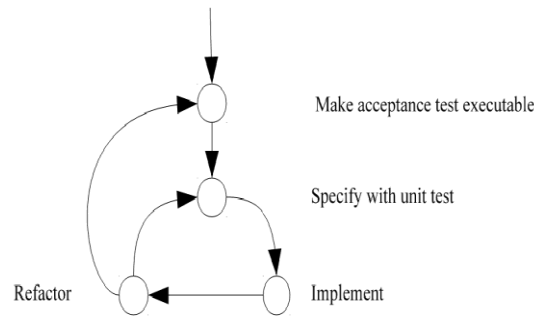


Fig. 3: BDD Implementation Cycle

III. . COMPARATIVE STUDY

So let us discuss about some difference between these terms and find out how are they different from each other.

- As discussed earlier, the immediate obvious difference between TDD and ATDD is the ‘A’. While that may sound sarcastic, the point is that TDD (as usually practiced) has an implied U on the front, standing for Unit, while the A stands for Acceptance. TDD focuses on the low level, ATDD on high level.

Sr. no	Method	Phases	Description
1	XP	Exploration	Write story for current iteration
		Iteration Panning	Prioritize Stories, effort and resource estimates
		Iteration to release	Analysis, design, coding, testing
		Production	Rigors testing,
		Maintenance	Customer supports, release for customer use
		Death Phase	No more requirements
2	Scrum	Pre-game	Preparation of product backlog list ,effort assessment, high level architectural design
		Development	Sprints, analysis, design, delivery,
		Post game	System testing, integration testing, documentation releases
3	FDD	Develop over all model	Scope, features, model, use cases are decided in various iterations
		Build the feature list	Feature list is prepared
		Plan by feature	Not clearly specified
		Design by feature	Not clearly specified
		Build by feature	
4	DSDM	Feasibility Study	Feasibility of the system is assessed
		Business Study	Essential business and technology characters are analyzed
		Functional model iteration	Analysis, functionality prioritization, nonfunctional requirements and risk assessment.
		Design and build iteration	Build and testing of system
		Implementation	Actual production of the system
5	ASD	Speculate	Project initiation, adaptive cycle planning
		Collaborate	Concurrent component eng.
		Learn	Review, F/A, Release

Table 1: Summary Of Software Development Of Life Cycle Of Popular Agile Methods

- So if ATDD leans towards the developer-focused side of things like [U]TDD does, the BDD is where the step of making it more customer-focused comes in. BDD is usually done in very English-like language, and often with further tools to make it easy for non-techies to understand. This allows much easier collaboration with non-techie stakeholders, than TDD.
- By contrast, TDD tools and techniques are usually much more techie in nature, requiring that you become familiar with the detailed object model (or in fact create the object model in the process, if doing true test-first canonical TDD). The typical non-programming executive stakeholder would be utterly lost trying to follow along with TDD, let alone participate, and frankly shouldn't be involved with that level of detail.

- BDD focuses on the behavioral aspect of the system rather than the implementation aspect of the system that TDD focuses on. BDD gives a clearer understanding as to what the system should do from the perspective of the developer and the customer. TDD only gives the developer an understanding of what the system should do.

IV. CONCLUSION

TDD and BDD practice follows the same basic idea as Acceptance Test-Driven Development, where acceptance tests are elaborated for each user story and automated as the software is built. BDD is more than TDD because it focuses on collaborating with business people. Dan North, who originated the BDD moniker, noticed that business people switched off in conversations about "tests" as these seem to be too technical. He hoped that framing conversations about "behaviours" would be a way to engage the whole team. BDD is about figuring out what to build that helps flesh out behaviour. BDD isn't an Agile framework or project management approach. Teams using Scrum or Kanban with BDD will need to figure out what to put in their backlogs and boards for planning purposes. You might want to measure the number of BDD scenarios delivered instead of velocity but often these scenarios are too fine-grain for release planning purposes.

REFERENCES

- [1] Divya Prakash Shrivastava, Unit test case design metrics in test driven development in IEEE eXplore, 2011 on Computer Technology, 1 – 6, ISBN: 978-1-4244-9795-9.
- [2] K. Beck, Extreme Programming Explained, Pearson Education Low price Edition Asia, 2006.
- [3] A. Cockburn, Agile Software Development, Pearson Education, Asia, Low Price Edition, 2007.
- [4] Divya Prakash Shrivastava, Rachapudi. V. Lakshminarayan, Sagaram V.S.L. Sujatha, (2010) New Engineering Technique of Software Development in IEEE eXplore, on Computer Technology and Development, 671-676, ISBN 978- 1-4244-8845-2/ 10
- [5] D. Janzen and H. Saiedian. Does Test-Driven Development Really Improve Software Design Quality?. IEEE Software. vol. 25, no. 2, 2008..
- [6] A. Gupta and P. Jalote. An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In Proc. of Empirical Software Engineering and Measurement , 2007, pp.285-294.
- [7] D. North, Introducing BDD, 2006. Available at: <http://dannorth.net/introducing-bdd>.
- [8] D. Chelmsky, D. Astels, Z. Dennis, A. Hellesoy, D. North. The RSpec book: Behaviour Driven Development with RSpec, cucumber and friends, Pragmatic Bookshelf, 2010.
- [9] D. Astels, A new look at test driven development, http://techblog.daveastels.com/files/BDD_Intro.pdf.