Volume 1 : Issue 4 [ISSN 2374-1619]

Publication Date: 27 December, 2014

Traditional Vs Agile Methodology: An Analysis on Challenges faced in Testing Perspective

Balaji Sundramurthy, M.C.A.
Faculty of Computing Sciences,
Gulf College, Sultanate of Oman

Ronald S. Cordova, Ph.D.
Faculty of Computing Sciences,
Gulf College, Sultanate of Oman

M. Sundara Rajan, Ph.D.
Department Of Computer Science,
Government Arts College, India

Abstract **- There are numerous methodologies available for developing and testing software. The methodology we choose depends on factors such as the nature of project, the project schedule, and resource availability. Although most software development projects involve periodic testing, some methodologies focus on getting the input from testing early in the cycle rather than waiting for input when a working model of the system is ready. Those methodologies that require early test involvement have several advantages, but also involve tradeoffs in terms of project management, schedule, customer interaction, budget, and communication among team members. Agile Testing is a set of timesaving techniques specifically designed to make the work of agile testing teams easier and more productive. It is an empowering process that produces great results and has a simple mission: Get the best possible testing results with the least amount of work. These challenge- and solution-based techniques do not require major changes in your existing workflow. We can adopt them in increments, which enables us to focus on one specific challenge and meet it head-on with a precise, targeted solution. Common challenges agile teams face and recommended solutions to handle them quickly and effectively .This paper discusses how testing fits and challenges into traditional/Agile methodology and then discusses the test-driven development practice in Agile Methodology in detail.**

Keywords-**Traditional methodology, Agile methodology, Test-Driven development**

I. Introduction

A Software Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, A number of software development life cycle (SDLC) models have been created: waterfall[11], spiral, V-Model [12], rapid prototyping, incremental, Agile model. Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, uses test driven development [2] and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed. By using test-driven development, we can ensure that each feature of an application block is rigorously tested early in the project life cycle. Early testing significantly would solve most of the issues that will be encountered by the development team and it will enable them to keep track of all the defects.

A. Key Differences between Traditional and Agile Methodology

- Development is incremental rather than sequential. Software is developed in incremental, rapid cycles. This results in small, incremental releases, with each release building on previous functionality. Each release is thoroughly tested, which ensures that all issues are addressed in the next iteration.
- People and interactions are emphasized, rather than processes and tools. Customers, developers, and testers constantly interact with each other. This interaction ensures that the tester is aware of the requirements for the features being developed during a particular iteration and can easily identify any discrepancy between the system and the requirements.
- Working software is the priority rather than detailed documentation. Agile methodologies rely on face-to-face communication and collaboration, with people working in pairs. Because of the extensive communication with customers and among team members, the project does not need a detailed requirements document.
- Customer collaboration is used, rather than contract negotiation. All agile projects include customers as a part of the team. When developers have questions about a requirement, they immediately get clarification from customers.
- Responding to change is done, rather than extensive planning. However, it suggests changing the plan to accommodate any changes in assumptions for the plan, rather than trying to follow the original plan.
- Agile testing was different in many ways from ‘traditional’ software testing. The biggest difference is that on an agile project, the entire development team takes responsibility for quality. This means the whole team is responsible for all software testing tasks, including acceptance test automation. When software testers and programmers work together, the approaches to test automation can be creative.

Volume 1 : Issue 4 [ISSN 2374-1619]

Publication Date: 27 December, 2014

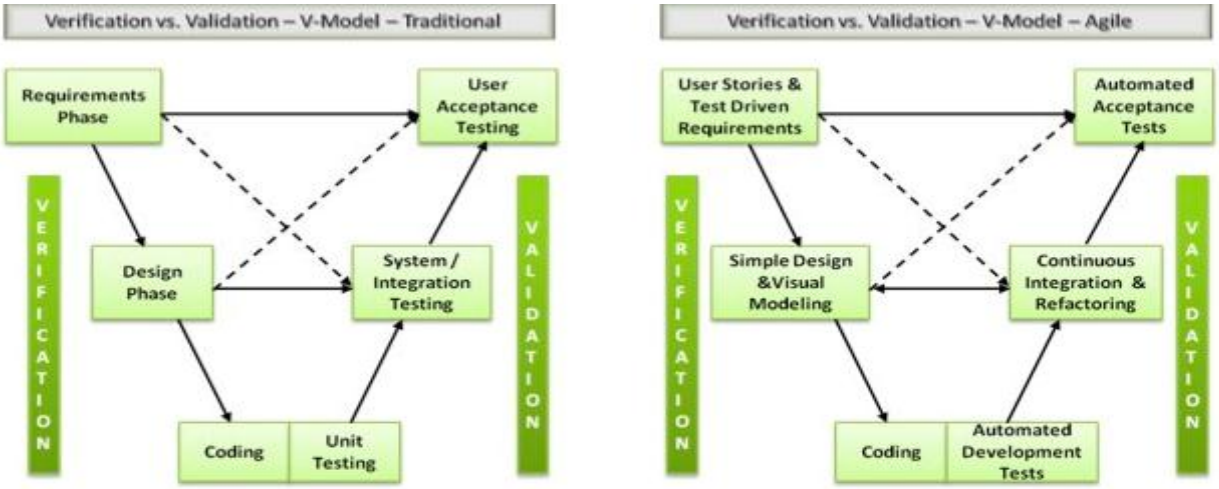

Fig 1: Testing Aspects of Agile V-model Vs traditional V-model

II. Challenges in Traditional Methodology

- Significant delays between when software is written and development receives feedback
- Defects found late in the process can have major implications when changed
- Changing business requirements affect test cases that have already been developed
- Communications create risk that different groups may have different expectations of the final product
- Quality suffers and many QA activities get left out when testing is the last activity before a fixed release date

III. Benefits of Agile Testing

- On-going feedback to developers allows testers to ask the right questions at the right time.
- Early identification of dependencies, technical or testing challenges and road blocks.
- Embraces change as a healthy and real part of software development.
- Team collaboration helps everyone work together toward a common goal.
- Quality comes first because final acceptance criteria are established prior to the work beginning.

IV. Analysis of Test Driven Model in Agile Methodology

Test-driven development [2] is one of the core practices of Extreme Programming. The practice extends the feedback approach, and requires that we develop test cases before we develop code. Developers develop functionality to pass the existing test cases. The test team then adds new test cases to test the existing functionality, and runs the entire test suite to ensure that the code fails (either because the existing functionality needs to be modified or because required functionality is not yet included). The developers then modify the functionality or create new functionality so that the code can withstand the failed test cases. This cycle continues until the test code passes all of the test cases that the team can create. The developers then refactor the functional code to remove any duplicate or dead code and make it more maintainable and extensible. Test-driven development reverses the traditional development process. Instead of writing functional code first and then testing it, the team writes the test code before the functional code. The team does this in very small steps—one test and a small amount of corresponding functional code at a time. The developers do not write code for new functionality until a test fails because some functionality are not present. Only when a test is in place do developers do the work required to ensure that the test cases in the test suite pass. In subsequent iterations, when the team has the updated code and another set of test cases, the code may break several existing tests as well as the new tests. The developers continue to develop or modify the functionality to pass all of the test cases. Test-driven development allows us to start with an unclear set of requirements and relies on the feedback loop between the developers and the customers for input on the requirements. The customer or a customer representative is the part of the core team and immediately provides feedback about the functionality. This practice ensures that the requirements evolve over the course of the project cycle. Testing before writing functional code ensures that the functional code addresses all of the requirements, without including unnecessary functionality. With test-driven

development, we do not need to have a well-defined architectural design before beginning the development phase, as we do with traditional development life cycle methodologies. Test-driven development allows us to tackle smaller problems first and then evolve the system as the requirements become clearer later in the project cycle.

A. Advantages

- Test-driven development [2] promotes loosely coupled and highly cohesive code, because the functionality is evolved in small steps. Each piece of the functionality needs to be self-sufficient in terms of the helper classes and the data that it acts on so that it can be successfully tested in isolation.
- The test suite acts as documentation for the functional specification of the final system.
- The system uses automated tests, which significantly reduce the time taken to retest the existing functionality for each new build of the system.
- When a test fails, we have a clear idea of the tasks that must perform to resolve the problem. Also have a clear measure of success when the test no longer fails. This increases the confidence that the system actually meets the customer requirements.

Test-driven development [2] helps ensure that source code is thoroughly unit tested. However, we still need to consider traditional testing techniques, such as functional testing, user acceptance testing, and system integration testing. Much of this testing can also be done early in our project. In fact, in Extreme Programming, the acceptance tests for a user task are specified by the project stakeholder(s) either before or in parallel to the code being written, giving stakeholders the confidence that the system meets their requirements.

v. Approaches Taken on Project: Agile Testing

- The test team and the development team were not formally separated. The developers worked in pairs, with one person developing the test cases and the other writing the functionality for the module.
- There was much more interaction among team members than there is when following a traditional development model. In addition to using the informal chat-and-develop mode, the team held a 30 minute daily stand up meeting, which gave team members a forum for asking questions and resolving problems, and weekly iterative review meetings to track the progress for each iterative cycle.
- Project development began without any formal design document. The specifications were in the form of user stories that were agreed upon by the team members. In the weekly iterative review meetings, team members planned how to complete these task and how many iterations to assign for each task.
- Each task was broken down into several tasks. All of the stories and corresponding tasks were written down on small cards that served as the only source of design documentation for the application block.
- While developing each task, NUnit test suites were written to drive the development of features.
- No formal test plans were developed. The testing was primarily based on the tasks or stories for feature development. The development team got immediate feedback from the test team. Having the test team create the quick start samples gave the development team a perspective on the real-life usage of the application block.
- After the task passed all of the NUnit test cases and was complete, quick start samples were developed to showcase the functionality. The quick start samples demonstrated the usage of the application block and were useful for further testing the code in the traditional way (functional and integration tests). Any discrepancies found in this stage were reported immediately and were fixed on a case-by-case basis. The modified code was tested again with the automated test suites and then was handed over to be tested again with the quick start samples.
- Traditional test phases typically fit with an agile testing approach
- Unit testing is still completed by developers as usual, but ideally there's a much stronger emphasis on automated testing at the code/unit level.
- In eXtreme Programming (XP) [8], there is also a strong emphasis on test driven development, which is the practice of writing tests before writing code. This can start simply with tests (or 'confirmations') being identified when a 'task' is written, and can go as far as actually writing automated unit tests before writing any code.
- System testing and integration testing are rolled together. As there is at least a daily build, and ideally continuous integration, features can be tested as they are developed, in an integrated environment. As per waterfall, this stage of testing is ideally carried out by professional testers. Importantly, each feature is tested as it's developed, not at the end of the sprint or iteration, and certainly not at the end of the project.
- Towards the end of each sprint, when all features for the iteration have been completed

Volume 1 : Issue 4 [ISSN 2374-1619]

Publication Date: 27 December, 2014

(i.e. developed and tested in an integrated environment), there needs to be time for a short regression test before releasing the software. Regression testing should be short because it is automated; test driven development, with features tested continuously in an integrated environment, and it should not result in many surprises.

- Finally, on a very large project, where a release must practically span multiple sprints to be of any value, a 'stabilisation' sprint may be worthwhile to make sure everything is okay before release. This should, however, be a short duration and the need for a stabilisation sprint should be avoided if at all possible, by trying to deliver releasable quality in each and every sprint along the way. If it is required, this sprint should be all about reducing any defects prior to launch, and the scope of development should at that time be frozen

VI. Tester Role in Traditional Model

- A Tester will receive a requirements document which he/she proceed to review.
- A Tester eventually gets a requirements document that is considered baseline or signed-off
- A Tester analyzes these requirements to create test conditions and test cases
- A Tester writes the test procedures
- A Tester then waits for a piece of software to deploy in test environment by developer.
- A Tester now starts executing the tests
- A Tester begins re-executing some of these tests as now starts iterating through new builds which are released to fix bugs or they may even include new functionality
- A Tester then reaches the acceptable risk and software is released

VII. Key Challenges for Tester in Agile Project

- No traditional style business requirements or functional specification documents. Tester will have small documents which only detail one feature. Any additional details about the feature are captured via collaborative meetings and discussions.

- A Tester will be testing as early as possible and run the tests continuously throughout the lifecycle so expect that the code won't be complete and is probably still being written.

- Acceptance Test cases are part of the requirements analysis process as tester are developing them before the software is developed

- The development team has a responsibility to create automated unit tests which can be run against the code every time a build is performed.

- With multiple code deliveries during the iteration, regression testing requirements have now significantly increased and without test automation support, Tester ability to maintain a consistent level of regression coverage will significantly decrease.

VIII. Processes Followed in Agile Testing

- The Customer prepares the Business Requirements and the Business Analyst or the Engineering team reviews it. Ideally, the Quality Assurance/Testing team is also involved in reviewing these requirements in order to be able to plan further stages accordingly.
- During the Design and Implementation stages, the Engineering team writes User Stories and the analysis of issues at various stages. The Customer reviews these on regular basis and updates the Requirement specifications accordingly. The Testing team would follow up on regular basis at every stage until a consolidated documentation is prepared. This is to ensure that the Customer, the Engineering team and the Testing team are at the same page always and thus ensuring complete test coverage.
- While the Engineering team starts the implementation, the Testing team starts with test planning, test strategies and test cases preparation. These would be properly documented and handed over to the Customer and the Engineering team for review. This is to ensure the complete test coverage and avoid unnecessary or redundant test cases.
- As and when the Developer implements the code, the Testing team identifies if the application can be built using this code for a quick testing. This is to identify the defects at the early stage so that the developer can fix them in the next round on priority basis and continue with further development. This iteration continues until the end of the code implementation. Once the testing cycle starts, the Test team can now focus more on major test items such as Integration, Usability Testing and System Testing

Volume 1 : Issue 4 [ISSN 2374-1619]

Publication Date: 27 December, 2014

IX. Processes Followed in Traditional Testing

- Receive requirements document from the customer then proceed to review
- Eventually will get requirements document that is considered baselined or signed-off
- Analyse these requirements to create test conditions and test cases
- Write test procedures
- Then wait for a piece of software to miraculously appear in test environment.
- Then now start executing tests
- Now begin re-executing some of these tests as you now start iterating through new builds which are released to fix bugs or they may even include new functionality
- Then reach the acceptable risk, enough testing point (or the fixed immovable deadline) and the software is released

X. Conclusion

In this paper, we described the different testing approaches to software development through traditional and agile methodologies [1]. Furthermore, we initially criticized on both traditional and agile methodologies followed by the comparison. Further, we discussed on benefits, analysis on test driven development & tester challenges in both traditional and agile methodology. The need for business to respond rapidly to the environment in an innovative, cost effective and efficient way is compelling the use of agile methods to developing software. The future of agile methodologies seems very dominant. In general, there are some aspects of software development project that can benefit from an agile testing approach and others can benefit from a more predictive traditional testing approach.

References

[1] Beck, Kent, et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. Retrieved on 7 April 2014, http://agilemanifesto.org .

[2] Beck, Kent (2002). "Test-Driven Development by Example". Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.

[3] Palmer, Steve & Felsing, Mac (2001). "A Practical Guide to Feature-Driven Development(1st ed.)". Pearson Education.

[4] Peterson, K. (2010a). "Doctoral research in Sweden Implementing Lean and Agile Software Development in Industry".

[5] Koskela, Lasse. (2007). "Test Driven: Practical TDD and Acceptance TDD for Java Developers". Manning Publications Co. Greenwich, CT, USA.

[6] Pettichord, Bret. (2002). "Agile Testing: What is it? Can it work?" Retrieved on 9 April 2014 from http://www.sasqag.org/pastmeetings/AgileTesting20021121. pdf.

[7] Hendrickson, Elisabeth (2008). "Agile Testing, Nine Principles and Six Concrete Practices for Testing on Agile Teams" Retrieved on 15 April 2014 from http://testobsessed.com/wp-content/uploads/2011/04/ AgileTestingOverview.pdf.

[8] Crispin, Lisa (2003). "XP Testing Without XP: Taking Advantage of Agile Testing Practices". Retrieved on 17 April 2014 from http://www.methodsandtools.com /archive.php?id=2.

[9] Highsmith, Jim and Cockburn, Alistair (2001). "Agile Software Development: The Business of Innovation". Addison-Wesley, 2001.

[10] Larman, Craig (2003). "Agile and Iterative Development: A Manager's Guide". Pearson Education.

[11]Weisert, Conrad (2003). "Waterfall Methodology: there's no such thing!". Retrieved on 18 April 2014 from http://idinews.com/waterfall.html.

[12] "Overview of the Activity Model of the V-Model"(2006). Retrieved on 20 April 2014 from http://v-modell.iabg.de/v-modell-xt-html-english/.

[13] Peterson, K. (2010b). "Doctoral research in Sweden Implementing Lean and Agile Software Development in Industry".

[14] North, Dan (2006). "Introducing Behaviour Driven Development". Retrieved on 21 April 2014 from http://dannorth.net/introducing-bdd/.

About Authors:

Mr. Balaji Sundramurthy is a Lecturer in Gulf College, Sultanate of Oman. He has published over 8 papers about Agile methodology and he has received STC Testing certificate. He is currently a PH.D student under the supervision of Dr. M.Sundara Rajan and his research is based on Agile Methodology.

Ronald Cordova received his PhD in Information Technology from Hannam University, South Korea in 2009. Presently, he is a Lecturer and Research Coordinator at Gulf College, Sultanate of Oman. He has authored and co-authored several scientific publications. His research interests are software engineering and Internet applications.

Dr. M. Sundara Rajan is working as Assistant Professor in Government Arts College, Nandanam, Chennai. He is an experienced educationist with over 20 years of experience in teaching. He encourages research and is currently supervising research work of many students especially in areas of Software Engineering, Data Mining and Warehousing, computer Networking and Natural Language Processing.