Conference Proceeding Issue Published in International Journal of Trend in Research and Development (IJTRD), ISSN: 2394-9333, www.ijtrd.com

Practices for achieving Effective Quality Assurance Teams: An ERA (Enhanced & Rebuilding Approach) in Agile

[1]A C Bharath and [2]Padma Priya,
[1]Test Management, Reliance Jio Infocomm., India
[2]Founder, Agile Bodhi, India

Abstract: In the current IT industry, Agile has gained rapid interest. Software development industry is exploring the utilization of Agile development approaches in current competitive environment because of its better correspondence and coordination, improved efficiency, productivity and quality applications. Agile methods are known to have built-in quality management systems. However, there are still challenges seen in the real-life scenarios within the software organizations that are transitioning from traditional method to Agile development methods. Different organizations are transforming their traditional software practices into the Agile model. Traditional techniques in Quality Assurance (QA) are report-based and highly relying on examining and inspection methods whereas the Agile Quality Assurance techniques are built-in daily activities by self-organized teams.

Research proposal here aims in studying various challenges faced in terms of assuring quality in Agile and focusing on providing a conceptual approach aiming to organize, analyse and make sense out of the dispersed field of Agile favouring project stakeholders to focus on the important changes required and the challenges involved in Quality Assurance teams following the Agile.

Keywords *– Agile Testing, Quality Assurance (QA), Test Automation, Test Driven Development (TDD), Test Commit Review (TCR).*

I. INTRODUCTION

Software development have been progressed since 1970s. Agile approaches came into existence post the necessity over a light way to do software development in order to oblige the changing requirement environment. Agile software development methodologies are recently gaining widespread popularity [1]. The Agile Manifesto [2] states valuing "individuals, and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan" [3].

Software testing is an essential and one of the key important parts for any application to ensure functional correctness as per customer requirements. Proper testing techniques must be adapted to suit the application so as to enhance the performance and ease the maintainability. Software testing is the process of evaluating a software application to identify the differences between a given condition and expected outcome assessing the feature of the inspected software application. Testing assesses the quality of the product assuring the quality of the application.

II. BACKGROUND AND RELATED WORK

Software testing is a process is performed during the development process for delivering better quality software. Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test [4]. In other words, software testing is a composite of the verification and validation processes. Verification is that the process to form sure the customers get satisfied with the conditions imposed at the beginning of the event phase. In other words, to make sure the product behaves the way one who needs it. Validation is that the process to form sure the merchandise satisfies the required requirements at the top of the event phase. In other words, to form sure the merchandise is made as per customer requirements. If software testing is carried out completely in the early stages in the product lifecycle, then software production will be less expensive in the long run period. Software bugs can be found at any moment of time in the product lifecycle.

A. Agile Testing

At the beginning of a every application or project development activities, the agile testing begins. It includes the seamless integration between testing and development activities. Traditionally, earlier days post after the coding process, the testing was followed as a separate task. But in agile, the testing is a continuous activity involving testers between product owners and developers. This creates an ongoing feedback loop which helps developers improve their code according to requirements. Agile testing has been adopted by enterprises who need continuous changes throughout the software development and testing lifecycle. Recently, it is found that the popularity of Agile methodologies is significantly increasing, which demonstrates the need for Agile testing processes and techniques to reflect changing testing practices and meeting demands [5].

In Agile Testing, the word 'Agile' noun refers and implies something to do very quickly. Hence, Agile Testing refers to validating the client requirements as soon and early as possible and make it customer friendly. As early as the build is out, testing activities are expected to get initiated and report the bugs quickly, if found any. As a Tester, one need to provide our thoughts on the client requirements rather than just being the audience at other end. Emphasis are being laid down on quality of deliverable in spite of

short timeframe, which will further assist in reducing the cost of development and feedbacks from Quality Assurance (QA) teams will be implemented in code avoiding the defects coming from the end users.

B. Key Characters of Agile Testing leading to Benefits

In current modern digital economy trends, it is often difficult or impossible to predict how a computer-based system application (E.g. A web-based or Mobile application) will evolve as time passes. Market dynamics are changing rapidly, the needs of end-users are changing and new competitive challenges are emerging without notice. You won't be able to completely identify specifications in certain cases until the project starts. One must be agile enough to respond to a fluid business environment. Fluidity implies change, and change is expensive, particularly when it is uncontrolled or poorly managed. There are a few key characters that exist in Agile testing. These include,

i. *Regular Communication with Stake Holders* - Testers interact with varoius stakeholders [6] to clearly establish and understand the business requirements and project expectations so that they can be helping developers comply with the product roadmap and meet customer needs and satisfaction. This can be helpful in delivering the application or software to the customer in a frequently defined timeline called 'Release'. Varoius studies and reports says that key attributes such as active stakeholder participation, self organizing teams, team size etc. have impact both on productivity and quality of the finished product [7]

ii. *Close interaction with Developers* – Test execution process is straight linked to the development process. Testers are also part of the development team, where they report on quality issues that can affect end-users or customers, and the interaction will suggest how to improve the solution and providing quality. Information derived post software testing may be used to correct the process by which application can be better developed [8].

iii. *Entire team is involved in quality assurance* – Whole team is passionate about quality; developers build unit test cases for a better testing process and to enhance the quality of audits. Meanwhile, in a practical way the testing activity can also be performed by non-dedicated software testers [9]. Developers can also follow the recommendations of testers for test requirements and code improvements. Agile testing is integrated from the beginning stage unlike the conventional methods. Hence, Agile testing becomes more accurate, reliable and time saving.

Following are the benefits of following the Agile testing [10] that every stake-holders getting benefitted

- Development and testing activities are concurrent
- Everyone works as a team towards a common goal and everyone is responsible for the quality
- Continuous integration and customer feedback
- Less risk of squeezed time period
- Working software is developed and delivered to the customer frequently

C. Persisting Challenges in Agile Testing

The very essence of Agile development is delivering working software frequently, whenever adding or enhancing alite features. which is useful to the customer. That itself possessing a lot of challenges, not only for testers, but also for developers and anyone else involved in the delivery of the application. There are various challenges [11] that are being faced and practiced on quality assurance activities. Continuous nature of Agile development processes raises below few serious testing challenges. These includes,

1. *Changing Requirements* - Changing requirements or dropping stories mid-sprint is not uncommon in Agile projects. This can be a nightmare for the whole team as it means that the work already carried out might be scrapped completely or changes should be made to what's already half done. These requirement changes and last-minute requests can affect the scope of testing which can frustrate testers. Try getting the developers involved in testing as well, as testing and quality should be the whole team responsibility. Studies tell us "Early involvement and the flexibility to adjust to frequent changes are the keys to successful quality assurance (QA) in an agile development environment" [12].

2. *Not Enough Information on the Story* - There will be times when a product owner who writes user stories, has some idea about a new feature but doesn't have all the details to write a good set of acceptance criteria to fully define the behavior of the feature. They ask the development team to create a prototype so they can get more ideas about the functionality and behavior of the feature. This creates a challenge for testers because there is a lack of understanding and requirements, so proper test cases can't be constructed.

3. *Continuous Testing* - In Agile, testing is not a phase, it's an activity. Testing starts from the very beginning, even before the development starts. In order to have a smooth ride during the sprint, the stories in the backlog should have been elaborated during the story grooming sessions. This means the QA should collaborate with product owners to learn the details of the story and then help write good acceptance criteria. Providing early feedback to developers is crucial and challenging for testers. As testers, not only we have to make sure that the new feature works as specified according to its acceptance criteria, we have to also make sure that the new code hasn't broken existing functionality, i.e. we haven't regressed, and we have to provide this information quickly.

4. *Technical & Automation Skills* - Working in an Agile environment, means that the testers should be technically competent to help the developers with Integration Testing and API Testing, as well as scripting UI automation checks with Selenium or similar tool. If the testers come from a purely manual or exploratory background, they will find it difficult to keep-up with the pace of delivery as they need to test on a continuous testing. Performance testing is also important particularly for

web-based applications, to ensure the application can sustain high load during peak times. If the company doesn't have a dedicated performance tester, it is expected the functional testers to also be involved in performance testing.

5. *Lack of Communication* - No matter how good the process is or how well the above items are carried out, if there is a lack of communication amongst the team members or with product owners, designers, etc., nothing will work. If communication between developers, testers and the product owners is lacking, Agile testing will simply not work.

Basically, the Agile teams today have no single measurement of quality, which they can use to optimize and plan testing efforts. There are numerous metrics like unit test code coverage, lines of code (LOC) and code complexity, but none of them provides a clear picture as to "where we stand" with quality at each point in a sprint or release – which areas of the product are working well, which are less stable and at higher risk of quality issues. In agile environment, Quality Assurance activities have to be therefore integrated within the team daily activities so that it will flow seamlessly and brings needed benefits in improving product quality [13]. Most Agile teams are flying blind, responding to production failures or bugs, but unable to proactively focus on product areas which have the biggest quality issues.

With these evolving challenges in this area, organizations are struggling to find the answers about the acceptability and maturity of agile quality assurance [14][15]. Quality attributes are the main themes of the product for which customer are looking for to insure the product values in market as working.

III. ENHANCED REBUILDING APPROACH IN AGILE

Approaches for enhance and rebuild the Agile testing are as proposed here. At first, few enhancements are recommended on agile test automation process. Later, the enhanced approach related to Test Driven Development (TDD) is proposed for better quality assurance process [16]. Basically, at the beginning of any project, one needs to start setting up the test environment, according to the requirement of the Application Under Test (AUT). A test environment is the fundamental key level infrastructure for all test activities associated with quality assurance. Test environment infrastructure is a setup of software and hardware for the testing teams to execute test cases. Setting up the right test environment ensures software testing success. Any flaws during this process may cause extra cost and extra time in the delivery of applications to clients.

Testing through through-out the span till Production ensures the quality assurance of the application. These testing activities vary throughout the lifecycle. Mainly, during Iteration 1 (refer fig 1) , one performs initial setup tasks. This includes identifying the people who will be on the external testing team, identifying and potentially installing testing tools.

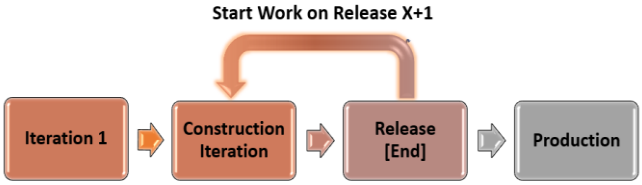

Figure 1. Environment & lifecycle through various Iterations

During construction iterations, a large amount of testing occurs - agilists test regularly test early, and typically test first. This is confirmatory checking against the current intent of the stakeholder and is usually unit-level milestone-based. This is a great start, but it's not the whole picture of research. The true aim should be to test, not to prepare to test, regardless of the style, and definitely not to write detailed documents on how one plans to test at some stage, hopefully. Agilists are still preparing and we are still writing paperwork, but our focus is on things of high importance, such as actual research. This objective highly achievable and can be fulfilled with process known as Agile Automation Testing.

A. Agile Automation Testing Process Enhancements

Agile Automation Testing in software is an approach of using test automation in agile. The purpose of agile automation testing is to make the application developing process more effective and efficient with preserving the quality and time in synchronization with proper resource utilization. Thus, the implementation of such a process requires a lot of coordination and collaboration between various resources and systems. Basically, every organization follows the automation testing process with their suitable applicable frameworks. Automation framework is focused and designed at component interface level, rather than validating only on the User interface (UI) level. The test automation test cases are executed for every modules of code integration builds before additional bring up tests which were executed manually.

Automation is a critical component to maintaining agility, and is a priority for the entire team through established practices/disciplines and a focus on continuous improvement. Continuous integration/builds, unit, functional & integration test execution and continuous/automated deployment are common examples of applying agile automation testing beyond the scope of traditional automated tests. In business terms, time is money. If one fails to accommodate automation in test execution process, the amount of time to execute the tests becomes high, this can be a major cause of challenges in Agile Testing as one would be spending a lot running these tests. Agile Test Automation bolsters quality assurance and quickens application delivery. Some of the core practices and guidelines for enhancing the agile test automation are recommended as follows:

1. *Focus on System Level:* While developing and ensuring the quality of the applications developed in agile environment, the user interface (UI) is bounded to experience many changes and multiple versions. With respect to the test coverage on UI maintenance, the overall efforts and time invested for automation may leads to time-consuming. Preserving the maintenance costs down and enhancing overall coverage, agile test automation is recommended to be focusing and conducted at the level of systems and services.

2. *Quality Test Scripts:* The next thing that automation script owners, typically the testers need to concentrate on is building the quality test scripts, in accounting and making flexibility on accommodating the regression testing cycles. It is recommended that the scripts should be built and maintained fulfilling the following quality criteria. This includes (a) Correctness, (b) Versioning (c) Maintainability (d) Portability (e) Integrity (f) Performance. The goal behind this is to perform smoother and accurate testing, in particular the regression testing without any intervention from testers. Team can cover the regression test cycle without making unnecessary adjustments when test scripts sounds quality and reliable, and can also benefit from efficiency, speed, and accuracy related advantages.

3. *Periodic Reviews on Test Scripts:* In order to recognize their validity for tests, it is also necessary to check the test data and cases from time to time. The periodic analysis and reviews of tests helps in recognizing the redundant and obsolete tests that are no longer applicable to the current test cycle. This will often help in minimizing costs and maintenance efforts. one can also validate and verify the content of tests by evaluating that are likely to have a long-term impact on the automation test scripts.

4. *Continuous Monitoring on Development Environment:* Monitoring the developers and the development environment constantly is often beneficial. The software development process consists of a network, such as virtual machines, cloud simulations, and external databases, from the back-end system architecture to front-end interactions. For instance, it is not mandate that a bug always required to come from the application; it may sometimes occur due to some sort of bad networks between devices, environment of creation, configurations, integrated system etc. Therefore, concentrating on the design and functionality of all the contexts associated with the development of an application is important. It will help the team to concentrate on quality of the product and product's consistency rather than scrambling through different data for identifying the leading cause of the bug. In agile development, preserving everything in place and an eye out for the processes always makes test automation more efficient and reliable.

5. *Maintaining Tests Lean & Tiny:* It is recommended that one need to make sure the test cases are lean and small. This would help to bind unnecessary test data to the test results, which would add less values. Holding smaller test cases can also help to make fast adjustments according to regression criteria. In addition, the maintenance of large test suites containing numerous codes, configurations, and scenarios is reduced, thus reducing the overall burden on the development environment. Lean helps one to discard something for the end user that does not create value. But how about test scripts and the dynamic process and infrastructure of Continuous Integration? Are they viewed as waste at Lean? Ideally, they should not. It is because they contribute directly to delivering on time what consumers require: quality applications. There are more and more projects in which development spends time in designing complex configurations, interfaces, hooks, etc. to activate the true power

   This can be explained with various instances. Many projects benefit from improving how logs are monitored, i.e. allowing live log display within the application or returning the log within the response of the web service. Here are some examples to consider. Also, for tools that allow [distributed] log indexing and analysis, the format of logs is important. GUI projects benefit from the development of testing hooks for particular sections or controls of the application. This helps test automation to design improvements at the UI level that are more robust. Teams often decide to change their API and create new techniques specifically for testing. The consumer, for example, does not request uninstall features, but it is really important for testing, so the team exposes this additional functionality explicitly for testing purposes.

6. *Not Automating Every Test:* Mostly, testers will attempt in automating each and every layer of an application and its components with the goal of achieving 100 percent coverage. However, since there can be millions of test variations, automating each and every test lead to higher costs and efforts. In addition, the team might also lose focus on significant tests when automating every task. Again, it is very likely that the experiments require manual intervention for such complicated situations, in which case, automation does not achieve the anticipated result. It is necessary to know When to automate and when to not automate tests. This holds true no matter what technique is used. Research says that, it shouldn't be automated if a test is done only once. "One-time tasks and exploratory testing/edge cases should not be automated. As per definition, the edge cases are typically one-off test cases, and generally it does not pay off the effort to automate it. Exploratory tests are best used to obtain knowledge of a new feature requirements and then tweak or revise tests based on the gained knowledge."

In addition to above enhanced rebuilding approaches the choice of automation tool, maintenance and auto-deployment of scripts with proper devops supports, time span for execution, Test execution reports can be considered in delivering robust, stable, high-quality applications.

B. Enhanced Approach on Test-driven development (TDD)

TDD or "Test-driven development" is an approach to code development [17][18] popularized by Extreme Programming, referring to a style of programming in which three activities are interrelated. This includes, a) coding, b) testing (in the form of writing unit tests) and c) design (in the form of refactoring). TDD can be simply described as the process where one writes a "single" unit test describing a unit outcome aspect of the program followed by executing the test, with the intention of failure as the program faces a lacking of that feature followed by implementing the simplest feasible "just enough" code, for marking the test as pass. In TDD, tests are written one at a time and by the same person developing the module [19].

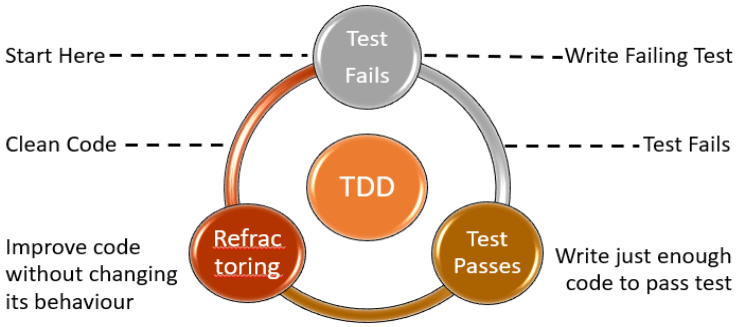

Figure 2. Test Driven Development

Later, "refactor" the code iteratively until it conforms to the simplicity criteria. Re-running all tests in merged code ensuing that refactored code does not break any previously passing test cases Do continue and following the same, with "accumulating" unit tests over time.

Benefits of TDD include that many teams report significant reductions in defect rates, at the cost of a moderate increase with respect to effects made on initial developments. TDD meant writing more tests and, in turn, programmers who wrote more tests tended to be more productive [20]. In an industrial case study that, with TDD, professional programmers were able to achieve a 40-50 percent improvement in code quality without any significant impact on their productivity [21]. Additionally, the same teams tend to report that these overheads are more than offset by a reduction in effort in projects' final phases. Although empirical research has so far failed to confirm this, veteran practitioners report that TDD leads to improved design qualities in the code, and more generally a higher degree of "internal" or technical quality, for instance improving the metrics of cohesion and coupling.

However, in few past experiments conducted earlier, it is observed that the developers follow a waterfall-like process, with all testing pushed to the end. As a consequence, most control subjects dropped testing altogether. The result was an apparent, short-term productivity disadvantage for the group that incurred the testing overhead [22]. TDD holds few typical individual mistakes, challenges, and gaps. This includes,

- Not executing the test infrequent basis
- Many testing checks on a single occasion
- Crafting the tests with very long procedures
- Devising overly trivial tests, for instance omitting assertions
- In addition to the above stated individual typical team pitfalls also persisting. This includes,
- Incomplete & partial adoptions in which only a few developers on the team uses the TDD.
- Test suite maintenance is not up to the mark and most commonly leading a test suite with a prohibitively long-running time abandoned test suite (i.e. seldom or never run) – sometimes as a result of poor maintenance, sometimes as a result of team turnover. Few challenges and gaps are persisting in TDD approach. Hence, in-order to enhance this - TCR is introduced

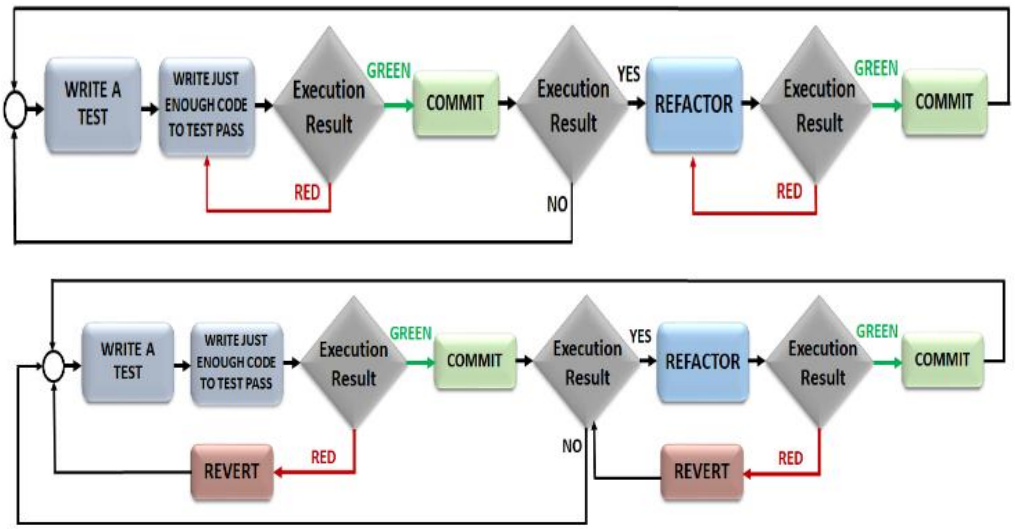

Figure 3 – TDD & TCR

1. *TCR Prototyped TDD:* TCR stands for "test && commit || revert", in which the ampersands and pipes are mandatory placeholders indicated in Unix-style command, which first executes the test-suite of the project. If the test results in green, the code gets committed. Otherwise, when the test results in red, the code gets reverted. In a simple algorithm the TCR can be expressed as like following pseudo code:

```
if testGreen ()
  commit;
else
  revert;
```

Figure 4 – Pseudo Code of TCR

TCR is a kind of new programming technique and its currently being trended as a successor to Test Driven Development (TDD) and the concept behind the TCR is extremely very simple. When your tests result in a successful outcome, developed code is committed. In case when your test resulting as failure behind any reasons, developed code is automatically about to get reverted back to the state where the tests results were last passed. The idea here is to make everything smaller - the amount of code you write between green test runs, your iterations around the test/dev/commit cycle and bigger problems broken down into smaller parts. Of course, writing tests is a given here because it is what the whole workflow is based on but they don't always have to come first.

While working with TCR, one would replace the test command which is already in use. As the code always gets committed when one executes your tests and they resulting in green. The revert-command, on the other hand, reverts the code to the last time your code was in a green state. This reverting leads to an interesting change in identifying how smart the code needs to be. Every mistake occurred while writing code leads to the commenting or deletion of the wrong code. This would be a quite frustrating one in the early stage. But, after some time, you change your behavior how you approaching the problems. You write a minimal, focused test, and just enough production code to make it pass. Then you refactor to a more general solution.

Considering this that TCR pushes one in the directions of the best practices, that one practices and follows from TDD: Creating small, focused tests, fake behavior until you can refactor towards a general solution and taking the baby steps towards the solution to problems and requisites. When one doing something else you increase the risk to lose your work. As the Test-Driven Development (TDD) is basically a programming practice [17][18], where you have to write test first and then write the code that will pass it. Finally doing a refactoring. Whereas, the Test Commit Revert (TCR) is an enhancement of the workflow. TCR is not TDD and there are differences to TDD.

With test-driven development, you start by writing with a failing Unit-Test which leaves you in a red state. As the second step, you fix this test by writing just enough production code to make all tests pass again, so you go into a Green state. Therefore, the Test-Driven Development is a Red-Green Workflow, at least regarding the state of the code. In contrast, TCR is a Green-Green Workflow. As soon as you go into a red state, TCR kicks in and bring you into a green state again. This, however, means, that you can't write a failing unit-test as you are used to in TDD. Instead, you have to write your failing unit-test AND write a fake implementation at once.

a. Benefits on practising Automation with TDD & TCR

Practicing the TCR brings back the excitement on programming with TDD. TCR pushes one into the direction of a very good. Doing baby steps, strive for a big test-coverage and so on.

The difference is that one should have discipline with TDD for practicing it. Especially in late hours, testers and developers gets indiscipline, skipping tests, using big unstable tests or dropping TDD as a whole, for forcing in free time.

With TCR, this does not happen, as the workflow constantly brings the best practices. As soon as one drop them, the code gets reverted. When one use them, you will not recognize TCR much. Hence one can say, that TCR is TDD with the discipline outsourced to a tool. One leads and reads a lot about discipline in TDD.

TCR pushes everyone to all the best practices you already know from TDD and don't require hard discipline. This alone is very simple and advantageous enough to use is. Especially because it doesn't change too much in practical usage after one got used to it.

Another major advantage is the effect on collaboration, that becomes possible. After every commit, which happens every few seconds, you can integrate with your team. Limbo and Code-Sync are the keywords here.

In case, when organizations are not performing or following with test automation, the overall test coverage might be low. But, as and when an organization implements the test automation, there is a sharp decline in the amount of time testers required for running different tests. Thus, it leads to accelerated outcomes and lowered business expenses. One can even implement automated browser testing to automate your browser testing efforts.

IV. CONCLUSION AND FUTURE WORKS

Agile methodologies came into existence after the necessity for a light-weight thanking to do software development so as to accommodate changing requirements environment. The aim of this paper is to attempt to make sense out of the enhanced rebuilding approached in Agile testing methods. Based on the observation of the analysis, practitioners are in a better position to understand the various properties of each method and make their judgment in a more informed way.

Scope for future works are vast and further, by evaluating critical links between various success factors in Agile industry and their impact on achieving high quality software, one can aim to propose quality assurance framework in Agile helping organizations to drive towards achieving excellence with improved software quality.

Acknowledgement

I would like to express my special thanks of gratitude to my Agile coach **Padma Priya Devarajan** for constant mentoring, guidance and encouragement for this research work. Also, I would like to convey thanks to **Prasad Saranathan** and **Harihara Prasad** for mentoring, reviewing and providing invaluable feedback and supports. Further, I am thankful to all my fellow researchers in ERA conference 2020, for their guidance giving inspiration in publishing this research work.

REFERENCES

[1] Scott W. Ambler, (2010) "Agile Adoption Rate Survey Results: February 2008" http://www.ambysoft.com/surveys/agileFebruary2008.html
[2] Agile Manifesto and Agile Principles, http://agilemanifesto.org/
[3] S. C. Misra, U. Kumar, V. Kumar and G. Grant,(2007) "The Organizational Changes Required and the Challenges Involved in Adopting Agile Methodologies in Traditional Software Development Organizations", 2006 1st International Conference on Digital Information Management, Bangalore, , pp. 25-28, doi: 10.1109/ICDIM.2007.369325.
[4] Kaner, Cem, "Exploratory Testing", (2006) Quality Assurance Institute Worldwide Annual Software Testing Conference. Orlando, FL.
[5] Gualtiero Bazzana, (2016) "Worldwide Software Testing Practices Report 2015 - 2016" ISTQB Effectiveness Survey Belgium.
[6] Tulasi Anand, (2008) "Stakeholders in Testing", Siemens SWI – Exchange of Experience- Testing.
[7] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, S. Z. Sarwar, (2010) "Agile Software Development: Impact on Productivity and Quality ", Proceedings of the 2010 IEEE ICMIT
[8] Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. ISBN 978-0-470-04212-0.
[9] Janet Gregory; Lisa Crispin (2009). Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley. pp. 23–39. ISBN 9780133749564
[10] Namratha, M. (2015). Benefits of Test Automation for Agile Testing. International Journal of Science and Research (IJSR)
[11] R. K. Gupta, P. Manikreddy and A. GV, (2016)"Challenges in Adapting Agile Testing in a Legacy Product," 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE), Irvine, Canada. pp. 104-108, doi: 10.1109/ICGSE.2016.21.
[12] Hewlett-Packard Development Company (2011) "The Impact Of Agile Development Processes on Quality Assurance", http://www.qcagileaccelerator.com/PDFs/Whitepapers/ImpactofAgileDev.pdf
[13] S. Bhasin, (2012) "Quality Assurance in Agile: A Study towards Achieving Excellence,"Agile India, Bengaluru, 2012, pp. 64-67, doi: 10.1109/AgileIndia.2012.18.
[14] P. McBreen, (2003) "Quality Assurance and Testing in Agile Projects", McBreen.Consulting.
[15] E. Mnkandla, and B. Dwolatzky, (2006) Defining Agile Software Quality Assurance. Proceedings of the International Conference on Software Engineering Advances (ICSEA'06)
[16] Lisa Crispin, (2002) coauthor "Testing Extreme Programming" Addison-Wesley publication. http://lisa.crispin.home.att.net
[17] Kent Beck, (2003) Test-Driven Development: by Example. Addison Wesley.
[18] Astels D. , (2003) Test Driven Development: A Practical Guide. Prentice Hall.
[19] R.C. Linger, H.D. Mills, and M. Dyer, (1987) "Cleanroom Software Engineering," IEEE Software.
[20] Erdogmus, Hakan; Morisio, Torchiano. (2005) "On the Effectiveness of Test-first Approach to Programming". Proceedings of the IEEE Transactions on Software Engineering, (NRC 47445). Archived from the original on 2014-12-22. Retrieved 2008-01-14.
[21] [George Williams] George B and Williams L,(2003) “An Initial Investigation of Test Driven Development in Industry,” Proc. ACM Symp. Applied Computing
[22] [Maximilien and L. Williams] E.M. Maximilien and L. Williams, (2003) "Assessing Test-Driven Development at IBM," Proc. International Conference Software Eng. (ICSE).