International Journal of Control Theory and Applications

ISSN : 0974–5572

© International Science Press

Volume 9 • Number 40 • 2016

A Proposal to Develop a Testing Framework for Agile Software Process

C.G. Anupama[a] Rashi Nair[b] and Soumi Roy[b]

[a]*Assistant Professor, Department of Software Engineering, SRM University*

E-mail: anupama.g@ktr.srmuniv.ac.in

[b]*Student, B.Tech. Software Engineering, SRM University*

E-mail: rashi_rakeshkumar@srmuniv.edu.in, soumi_roy@srmuniv.edu.in

Abstract: Agile software process is used by numerous organizations for developing their projects. This software process is increasingly in demand because of its ability to deal with changing requirements. This has proven to be a boon since the market is ever changing and because of it the customer's demands are ever changing. But this very boon of the process makes testing in these projects fairly tough and intricate. Ever changing requirements and development in iterations makes it unclear as to which type of testing should be performed when and how many test cases are relevant after the changes made. This paper suggests a simple framework as a proposal for different testing (unit, regression, system and acceptance) to be employed in each development iteration suitable for Agile Software Process.

Keywords: *Scrum, Agile, Usability Testing, Software Testing.*

1. INTRODUCTION

Agile methodology is an iterative, time boxed approach to deliver software incrementally from the start of the project, instead of delivering it all at once near the end [22]. It breaks down the project into little bits of user functionality called user stories, then Prioritize them and then continuously delivering them in short two to four week cycles called iterations. Agile methodology's major focus is to satisfy the customer's demands; therefore it always welcomes customer's ever-changing requirements. And it believes in practical demonstration of the product rather than the documentation [22]. There are some noteworthy agile methodologies - Scrum, Extreme programming (XP), Unified process (UP), Crystal, Feature driven development, etc.

1.1. Common Problems in Agile Development

1. **Scrum deviates from the real work of the project:** Scrum is the regular meeting of the project members with a Scrum Master. In a Scrum meeting, features and requirements are discussed; sometimes they come up with a new idea and discuss about that, which is good, but there are many times when their idea is a total deviation from the main project [15].

2. **Developers may feel that Scrum is unnecessary and slows them down:** Scrum Methodology is used by many companies for their software development. It gives the developers a way to sit down and discuss about any issues they are facing in the development of the project or if they have a new approach to continue the project in a more efficient way [18].

   On the other hand, if they are developing a project which already has a specific approach or a specific design to meet, then a collaborative approach like Scrum is not necessary.

3. **Generation of Inadequate Test Coverage:** In Agile Methodology customers requirements are ever-changing; therefore it doesn't have a proper documentation of any requirements, design or architecture; this becomes an issue while testing [18]. Sometimes it's not known what features should be tested first, what should be the testing sequence, how the testing should be approached like, which testing should be done first and how, etc. All these issues make the testing process very difficult.

4. **Performance Congestion:** As the requirements change the source code changes too. If the developer is not focused, he might not be clear about the change and may not be sure which part should be changed or altered; a few inefficient changes can increase the lines of code which will make the code more complex and lead to performance issues [22].

2. LITERATURE SURVEY

2.1. Exploratory Software Testing in Agile Project [1]

Exploratory testing uses the 'explore and find' approach to verify the functionalities and detect bugs in the system. This method doesn't need a lot of time to create the test plans (compared to the script-based methods) and hence is suitable for agile projects. The objects in exploratory software testing are – user input, state and user data.

2.2. Agile EDI Framework for B2B Applications [2]

A refined Agile Unified Process framework tailored for EDI (Electronic Data Interchange) is proposed consisting of the four phases of Unified Process along with custom steps to be followed in each phase:

1. **Inception:** Define, plan, prepare, feasibility study and contract
2. **Elaboration:** Identify, validate, evolve and staffing
3. **Construction:** Initiate, build, connect, map, test and finalize documentation
4. **Transition:** System test, user acceptance test, deployment, transfer and close

2.3. Applying Usability Testing to Improving Scrum Methodology in Develop Assistant Information System [3]

This study evaluates whether integrating usability testing with Scrum will improve software development related to testing, feedback and product quality. They get the users' feedback at the end of each sprint. Usability scenario and PSSUQ (Post-Study System Usability Questionnaire) are the main mechanisms used to support the usability-testing strategy.

3. PROPOSED FRAMEWORK

The proposed framework (Figure 1) is meant to serve as a guideline for agile teams to follow for testing.

3.1. Project initiation

This is the set of tasks performed before starting the product development [23]. It includes forming a team, briefing the team, setting up the first meeting with the customer, collecting requirements from the customer in the form of stories, extracting direct requirements from these stories and preparing a basic system design so that development can start.

3.2. Iteration 1

Get initial requirements : From the set of already extracted requirements (Let's call the set of all initial requirements as IReq), select the ones which signify core functionalities [21] - the requirements which would define the product. These are chosen so that in later iterations the team can build around these core functionalities.

Develop: This is when the first set of requirements is developed.

Unit test: Each unit in the part developed product is tested for its quality and output. Unit testing can usually be automated as smaller modules generally have basic functions [26]. But depending upon the product under development the team can also go for manual testing. The test cases for each module need to be documented for future use in development. Once all modules successfully pass the unit test, integrate them.

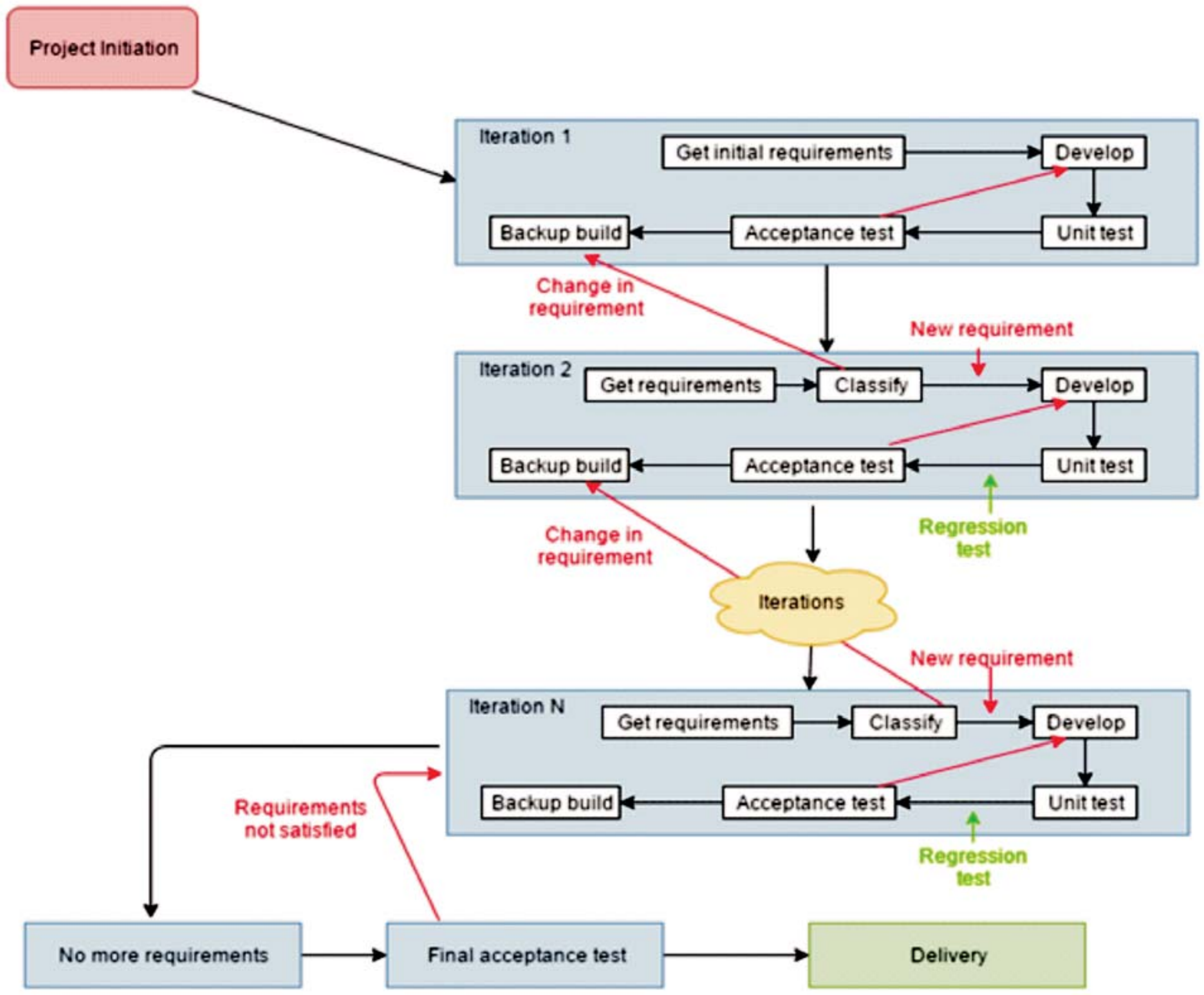

Figure 1: Proposed Framework

If in case time is running short, eliminate a few more requirements and develop the remaining as delivery of working software is the prime concern.

(Let's call the set of all implemented requirements as *1req* for iteration one)

Acceptance test: When the first sprint is at its deadline, a meeting is conducted along with the customer. The first increment of the product is showcased in this meeting. The main objective is to validate whether the requirements given initially are satisfied or not [26]. It is not necessary to deliver all the initial requirements. Just the prioritized ones would be enough. The reasonsbeing that the customer might change his mind and that delivering fewer requirements in each iteration thrives quality.

Hence, during the meeting the customer gives his feedback as to whether the delivered requirements are as per his wishes or not. He also tells whether there are any other requirements or changes.

The main results of this test are:

Sreq : List of previous requirements that need to be scrapped off (if any).

Nreq : List of new requirements or changes (if any).

Sreq and Nreq for iteration one will be referred to as 1Sreq and 1Nreq.

These modifications will be implemented in the next iteration.

Backup build : This step is a safety measure and might prove to be useful in further increments.

There's a chance that the customer might say a requirement is not needed anymore and then at later iterations he might say it's needed again [13]. In such cases, the developers can trace back to the particular build which implemented that requirement. This will help save time and effort. The test cases and coverages need to be backed up as well. Not necessary in a detailed manner, but in a way that it makes deriving test cases for the coming increments easier.

3.3. Iteration 2

Get requirements: Just like the previous iteration, this iteration also stars by sorting the requirements [22].

The total requirements at this stage are:

$$TReq \;=\; (((\mathrm{IReq} - \mathrm{1req}) - \mathrm{1SReq}) + \mathrm{1NReq})$$

From these requirements some are chosen for this iteration depending on the feasibility and suitability. Starting with these chosen requirements, the next step will begin.

Classify : These chosen requirements need to be classified as-

Change in previous requirement: Sometimes, the customer asks for modifications or engagements in previously stated requirements. Mostly the customer does not state explicit whether it's a change or not. The customer will give stories and the team needs to extract the requirements [22]. After extracting, the team must only identify if it's a change in any previously stated requirement or not.

New requirement : There's also a good chance that the customer has a completely new requirement. This must also be identified by the team [22]. It is usually eat to identify new requirements.

If the requirement is a change in a previous requirement then the team can refer/trace back to the required previous build and make modifications in the required module. This module can then be integrated into the current build.

If the requirement is a new requirement, the team can move on to the next step which is development.

Develop : Now it's time for development again. The team tries to implement the chosen set of requirements for this iteration. If, like the previous iteration, some requirements are not feasible or suitable currently then they are left to be implemented in later iterations.

Unit Test : New modules and modules in which changes have been made must be mandatorily unit tested [27]. If the team has ample time then they can test the other modules as well. Otherwise, a system test can be performed after integrating the modules.

(Let's call the set of all implemented requirements as *2req* for iteration two)

Regression test : From the second iteration onwards, regression test is a new step in this framework. The train being the fact that in regression test is done in order to identify whether any changes that have been made, introduced new bugs/issues in the software. Now, in the first iteration, the first set of requirements are implemented and only after implemented the customer feedback is taken so there's no chance of a major change. But in the second iteration, the changes requested in the acceptance test of the first iteration must be implemented (It's not necessary that all the requested changes will be delivered in this iteration only; they can be delivered in later iterations as well).

Therefore to check if any new bugs or issues have been introduced in the software, regression test must be done [27].

Acceptance test: Again, like the previous iteration, acceptance test is done to validate the implemented requirements [26]. At the end of the sprint, a meeting asking with the customer is held and same procedure is followed as in the previous iteration. The output of this meeting is again the list of requirements to be scrapped off and the list of new requirements/changes in requirements. For the second iteration they're referred to as *2SReq* and *2NReq*.

So the total requirements while starting the next iteration would be:

$$\text{TReq} = (((\text{TReq} - \text{2req}) - \text{2SReq}) + \text{2NReq})$$

(In further iterations, they will be called *nSReq* and *nNReq* where *n* represents the iteration number)

Backup build: This is again done as a safety measure if in case the customer wants a requirement that was previously implemented.

These steps are followed for n iterations depending on the customer's satisfaction. When customer says that there are ***no more requirements*** [10] and that he's satisfied with the product features then the iterations can come to an end.

After it is confirmed that the customer has no more requirements, the final product undergoes a system test. This must be done in order to tackle any residual bugs or issues that might exist in the product.

After the successful system testing, a ***final acceptance test*** [27] is performed. In most cases, the customer will be satisfied by now. But there's a chance he comes up with more changes. In that case, development process is continued from the last iteration into a new sprint. And at the end of this sprint, the customer validates the product again.

Finally when the customer is completely satisfied with the product, it is ***delivered*** [10] to them and the other proceedings of payment, etc. are followed. The product then goes into maintenance from then on.

As the agile methodology focuses on faster delivery of working product, exploratory testing [1] can prove to be helpful in this regard. It gives the tester freedom to interact with the system in any manner he wants without having to create detailed and exhaustive test plans. This is not mandatory of course. If a firm does not have time to invest in elaborate test plans, then they can go for the exploratory testing approach [24]. It will give them good results and reduce time required to carry out the tests.

4. CONCLUSION

Agile methodology is a powerful technique when it comes to customer satisfaction. To ensure that the testing procedure is not ambiguous and chaotic, agile development teams can follow this framework as a roadmap. The proposed framework suggests carrying out unit test and acceptance test in the first iteration. And from the second iteration onwards it includes regression test as an intermediate between the two to check for new issues. It also suggests backing up each build for future references and use.

Depending on the firm's need and suitability, exploratory testing approach can also be employed to save time and effort.

5. REFERENCES

[1] Exploratory Software Testing in Agile Project: http://ieeexplore.ieee.org/document/7219581/

[2] Agile EDI framework for B2B applications: httpeexplore.ieee.org/document/5328107/

[3] Applying Usability Testing to Improve Scrum Methodology in developing Assistant Information System: http://ieeexplore.ieee.org/document/7858222/

[4] GRAFT: Generic and Reusable Automation framework for Agile testing: http://ieeexplore.ieee.org/document/6949235/

[5] Performance Testing on an Agile Project: http://ieeexplore.ieee.org/document/4293621/

[6] Agile software testing in a large-scale project: http://ieeexplore.ieee.org/document/1657936/

[7] Testing of Changing Requirement in an Agile Environment - A Case Study of Telecom Project: http://ieeexplore.ieee.org/document/4344111/

[8] Adding usability testing to an agile project: http://ieeexplore.ieee.org/document/1667591/

[9] Case study on Critical Success Factors of agile software process improvement: http://ieeexplore.ieee.org/document/5917014/

[10] Agile Software Process model: http://ieeexplore.ieee.org/document/625042/

[11] Agile Software Process and its experience: http://ieeexplore.ieee.org/document/671097/

[12] Integrating testing into Agile software development processes: http://ieeexplore.ieee.org/document/7018519/

[13] Experiment Report on the Implementation of Agile Testing: http://ieeexplore.ieee.org/document/7113577/

[14] Agile Testing: A Systematic Mapping across Three Conferences: Understanding Agile Testing in the XP/Agile Universe, Agile, and XP Conferences: http://ieeexplore.ieee.org/document/6612876/

[15] Agile Testing: Past, Present, and Future -- Charting a Systematic Map of Testing in Agile Software Development: http://ieeexplore.ieee.org/document/6298092/

[16] Enabling Agile Testing through Continuous Integration: http://ieeexplore.ieee.org/document/5261055/

[17] Applying agile practices to avoid chaos in User Acceptance Testing: A case study: http://ieeexplore.ieee.org/document/7480122/

[18] Challenges in Adapting Agile Testing in a Legacy Product: http://ieeexplore.ieee.org/document/7577426/

[19] A preliminary analysis of various testing techniques in Agile development - a systematic literature review: http://ieeexplore.ieee.org/document/7783283/

[20] Independent Security Testing on Agile Software Development: A Case Study in Software Company: http://ieeexplore.ieee.org/document/7299961/

[21] Roger S, "Software Engineering – A Practitioner's Approach", Seventh edition, Pressman, 2010

[22] Craig Larman, "Agile and Iterative Development – A Manager's Guide", Pearson Education – 2004

[23] Ramesh Gopalaswamy, "Managing Global Software Projects", Tata McGraw Hill

[24] James A. Whittaker, "Exploratory Software Testing: Tips, Tricks, Tours, And Techniques To Guide Test Design", 2009

[25] Lisa Crispin, Janet Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams", Pearson Education, 2008

[26] Paul Ammann, Jeff Offutt, "Introduction to Software Testing", Cambridge University Press, 2008

[27] Srinivasan Desikan, Gopalaswamy Ramesh, "Software Testing: Principles and Practices", Pearson, 2012