Optimized Order of Software Testing Techniques in Agile Process – A Systematic Approach

Farrukh Latif Butt
Department of Software Engineering
Bahria University Islamabad,
Pakistan

Shahid Nazir Bhatti
Department of Software Engineering
Bahria University Islamabad,
Pakistan

Sohail Sarwar
Department of Computing and Technology
Iqra University Islamabad, Pakistan

Amr Mohsen Jadi
Department of CSSE
University of Hail, Hail, KSA

Abdul Saboor
Department of Software Engineering
International Islamic University Islamabad, Pakistan

***Abstract*—The designing, development of a software product needs lot of efforts whereas software testing is also a very challenging task but it is equally mandatory activity in order to ensure the quality of the product before shipping to customer. When it comes to the Agile model under which software builds are developed very frequently and development goes on a very high pace, software testing becomes more important and critical. Organizations following the agile methodology, encounter number of problems in formulating a software testing process unless they come up with a systematic testing approach based on right testing technique at a proper stage of the agile process. This paper addresses the relevant software testing techniques feasible at different stages of the agile process and proposes a dedicated software testing framework producing quality software products developed under agile methodology.**

***Keywords*—Agile methodology; software testing techniques; software build; software quality**

I. INTRODUCTION

Customers are demanding rapidly developed software products which is why organizations are shifting over agile methodology to deliver quality applications in short span of time [11]. The encouraging results of appropriate testing approaches in agile are making these software testing techniques more popular. In [2] authors highlights the need of automated software testing to better measure the quality of applications to be delivered to different industries. In addition to recognizing the need of automated testing, an automation framework has also been presented.

The quality assurance and testing activities add significant cost to the project which asks for the rational management and allocation of testing resources. Authors in [13], emphasizes on automated testing strategy to certify repeatable tasks through available tools. The stable and less error prone areas and features of a software product are good candidates for automated software testing. In agile process, software builds are provided to testing teams in a tight schedule that naturally creates pressure where testers have to cope sensibly with limited resources in terms of time and cost. The very first testing technique in this scenario is smoke testing that takes very small amount of time to assess the health of the build and results are communicated to whole team like whether this alpha build appears fine to continue for further use or not [4]. On the other hand, software developers implement user stories accommodating them in the software application that they certify at their own through writing unit tests against every user story or bug they fix that eventually make a library of unit tests [15]. On the availability of next build, software testers also assume the responsibility of regression testing to know whether fixing of bugs has ripple effects on other areas of the product or not? This aspect of regression testing has been elaborated in [6][17].

Once a release cycle goes through all the succession of iterations in agile process and reaches to the milestone of delivery, the Release Readiness Review (RRR) criteria is assessed before shipping the product. The research work [8], proposes a checklist for evaluating all the mandatory and relevant aspects for releasing a quality product and concerning responsible authorities sign off the checklist.

This paper proposes an optimized combination of testing strategies considering the appropriate techniques at right stage of the agile methodology for developing and delivering a quality product. The rest of the paper has following section: Section II provides the literature review based on the existing research in this domain. Section III proposes the methodology based on the efficient order of software testing strategies. Section IV presents results whereas section V concludes the research and outlines future work.

II. METHODS AND MATERIALS

The authors [1] proposes a software testing process dedicated to agile process which is based on a particular order of testing techniques with an intent of achieving more accurate and reliable results. They have presented an algorithm that minimizes cost and time of software testing phase as well as brings better results in terms of software quality.

In the execution of smoke test plan, automated software testing plays important role in replicating full length coverage with reduced sample size achieving reliable results and saving time and cost for other useful testing activities [10]. The authors make twofold research contribution [3], offering study

on agile testing process comprehensively and, on the other hand, provides useful documentation for engineers interested in extending software test framework specialized in agile model. The researchers suggest complete automation software testing process instead of manual certification of a software product compelling test engineers for irrelevant changes in the application. Moreover, in this age of industrial competition, automated software testing has become almost a must-do practice [2].

Authors emphasizes agility in the software testing process which, in addition to meeting user's requirements, improves throughput of software delivery and development process and minimizes the overall time of release cycle [12].

The validation of software product through unit testing before performing integration testing improvises the success possibility while working in agile. The research effort has been validated in five different projects deriving positive results [12].

Regression testing technique is very useful in validating the functionality of system after making modifications. There are different techniques to conduct regression testing however [6] used control graph based technique to assess the quality of the software when changes are made.

The verification of release readiness becomes vital to software quality when a sensitive system like JPL is under test. The goal of release readiness review is to assess the quality of the product with reference to any risks involved in delivery of product [8].

III. Methodology

It is presented that agile methodology for software development works on iterative philosophy in iterations one after the other [5]. The work done is reviewed in daily scrum meet ups and the progress is reviewed at the end of each iteration anyway. Thus, the quality assurance team has an opportunity to be indulged in the project right from the day one which asks for the formulation of a testing framework based on different software testing strategies. The testing framework in form of a combination of various practical testing approaches has been presented below.

A. Smoke Testing

In agile methodology, it is portrayed in [4] as soon as an alpha build gets handed over to testing team, initial round of testing is conducted to reveal bugs or problems in that software build. The objectives of the smoke testing are to test the basic features of the application; if they appear fine then testing team communicates smoke test results to the whole project team. One of the primary goals of performing smoke test is to save the time consumed on detailed testing in case the build is not stable and cannot be used further. Smoke testing is mainly done manually whereas there is possibility of doing the same with automation.

1) Manual Smoke Testing

Once the build is ready, it is released to QA, which takes into account the high priority test cases to find the critical bugs in the system. If the build fails, it is floated back to development end. Manual smoke tests are optimal if we have frequent changing product functionalities.

2) Automated Smoke Testing

If we have a stable version of product where major functionalities are not changing and there is high frequency of builds, then it is better to design the automated smoke tests. Each time the build is delivered, we just run the same automated smoke test to assess stability of build for further testing. Fig. 1 shows how smoke testing is carried out.

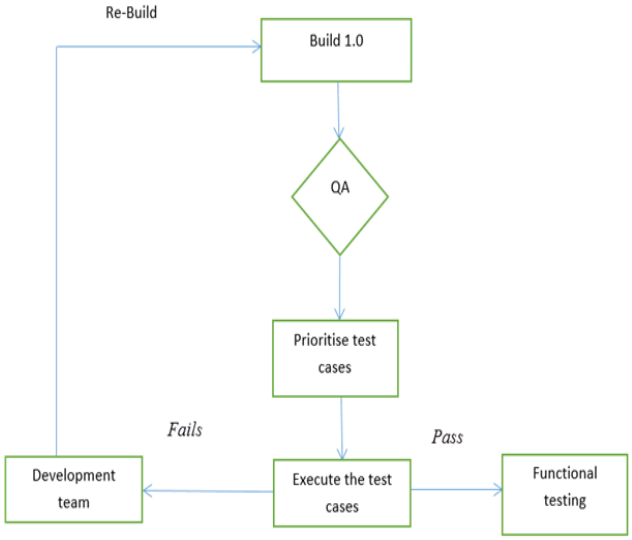

Fig. 1. Smoke testing process

B. Regression Testing

The defect fixing is the process of removing issues or problems reported in previous or older builds, once the defects are fixed they should not cause any ripple effects on other or same areas of the product. Regression testing expressed in [14], that ensures the changes committed to fix the identified bugs work fine and they have not introduced any side effects. The reduction of test suite is also a potential advantage offered by regression testing.

1) Reduction of Test Suite

The objective of reduction of test suite is to find out duplicate tests and to minimize the length of test plan by excluding the duplicates. Certainly, the assumption here is that individual requirement can be met by a particular test case. The Fig. 2 below gives an idea of identification of redundant test cases. On the horizontal axis requirements have been denoted by r while test cases are represented by t along y axis. We can learn from this figure that the goal of test case t1 can be achieved by selecting and executing merely test cases t2, t3 and t4. This way we can mark test case t1 redundant and eventually eliminate it from the test suite.

Test Case	Testing Requirements					
	r_1	r_2	r_3	r_4	r_5	r_6
t_1	x	x	x			
t_2	x			x		
t_3		x			x	
t_4			x			x
t_5					x	

Fig. 2. Identifying duplicate tests

2) Change-Based Method

The change-based method divides the system under test into different entities and observes the execution of tests to figure out the connection between tests and the entities of the program they run. The change-based method also categorizes the modified version of the program into different entities and finds out the entities which are changed by the original version of the program. This way all the tests that run entities of the changed version need to be re-run. Finally, any tests that run changed functions will be eventually shortlisted.

C. Unit Testing

For the sake of testing individual units of a software product, [7] recommends testing the smaller units of an application individually before they are collectively merged to form the whole product. Unit testing is typically performed by programmers or software developers though software testers can also conduct this testing.

1) NUnit test tool

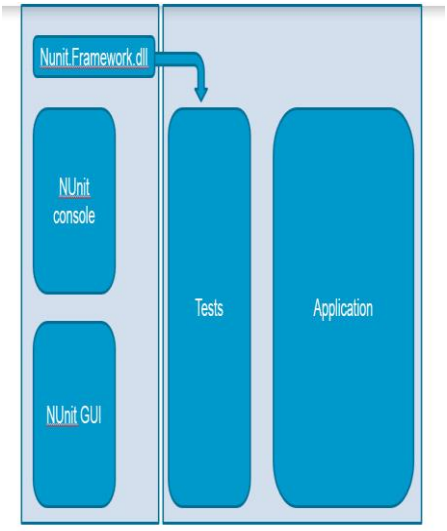

Fig. 3. Architecture of NUnit tool

NUnit is a tool for performing unit testing for Microsoft .Net technologies. This is an open source tool and serves the same purpose as JUnit does for Java. NUnit tool is based on the xUnit architecture that we will discuss later. It might be worthy to mention here that NUnit is neither an automated GUI tester nor a tool for scripting rather it is a unit or Application Program Interface (API) testing tool. Fig. 3 below demonstrates the architecture NUnit tool is based on for testing the underlying system.

2) xUnit Architecture

The NUnit tool is based on xUnit family of architectures which is specialized in providing basis for unit or API testing. Fig. 4 provides an overview of xUnit architecture.

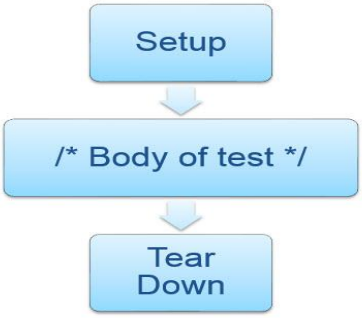

Fig. 4. xUnit design

3) Writing unit test in NUnit

A unit test is written in NUnit test tool in a test project that refers to Dynamic Linked Library (DLL) an API is based on. Also, the framework of NUnit tool must have been configured in the test project. The code snippet below illustrates a sample test written in NUnit as an example:

```
    [SetUp]
  public void test_Setup()
      {
      n = new int[3] { 2, 4, 6};
           i = new int[10] {3456, 5667, 76890, 67689, 64530,
          65789, 6758926, 64548903, 6476589, 63535885,};
      }

  [TearDown]
  public void test_CleanUp()
      {
      n = new int[3] { 0, 0, 0 };
      }

          // A = (a1, a2, a3) and n = length of A
      // A.M = (a1 + a2 + a3) / n

  [Test]
      [Category("ValidCases")]
      public void Test_ArithmeticMean()
      {
      int total = 0;
      foreach (int a in n)
      total = total + a;
      total = total / n.Length;
Assert.AreEqual(total, objMath.ArithmeticMean(n));
}
```

The execution of test suite in NUnit compiles results that can be exported for customization purpose for instance, the XML report. Fig. 5 below let's get an idea of the Automated Test Plans (ATP) executed using NUnit. In the scenario below, there are 9 tests in the test assembly loaded in NUnit GUI. The tests have been run to know the correctness of underlying API that performs arithmetic calculation. Intentionally, all of the flavors of tests results like passed, failed and ignored have been catered to better brief the execution. The passed tests nodes appear in green, failing are highlighted in red while ignored are yellow.

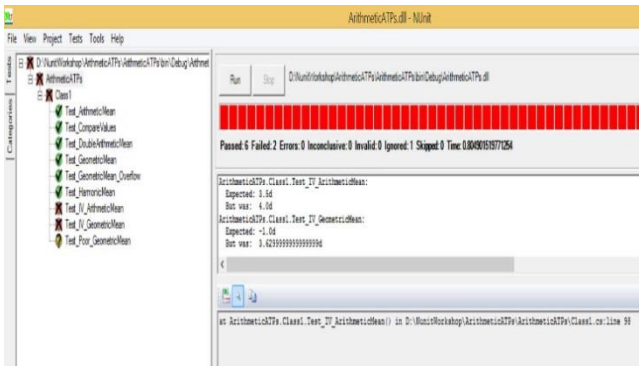

Fig. 5. Tests execution in NUnit

D. Automated Testing

Software automated testing has proved to be very handy in the field of software testing where test engineers can unhide flaws in the application and report them using automated testing tools and computer systems. The two basic aspects like application program interfaces and user interfaces which are the ideal candidate areas in a software product for automated testing [16]. Not necessarily all the components and functional areas must be considered for automated testing, rather it's the job of a test manager to decide which parts of the product should be considered for automated testing and which for manual or other testing strategies. The code coverage is measured through automated testing tools, however the effectiveness of faults detection on the basis of scripted unit tests has been demonstrated in [9].

1) Automation Process

The automation process can be commenced the moment requirements specification gets formalized. Fig 6. depicts automated testing process ranging from requirements specification through final report and deliverables. The specification of requirements provides basis to examine needs of end user as well as sets direction for software developers and test engineers. The test template can be used as a container for methods or areas to be tested through automated scripts. The script writers may check in their contents in the test template. The preliminary investigation of the system under test through automated testing reveals bugs or issues which are fixed eventually. The script or code in automatic software correction template keeps on updating depending upon the fix or changes committed to it. Finally, the summary based on the execution of all automated test plans, test cases, bugs identified, failed test cases etc. is generated in form of test report. On the other hand, at the same level, all or partial stuff involved in automated testing activity is presented as a deliverable.

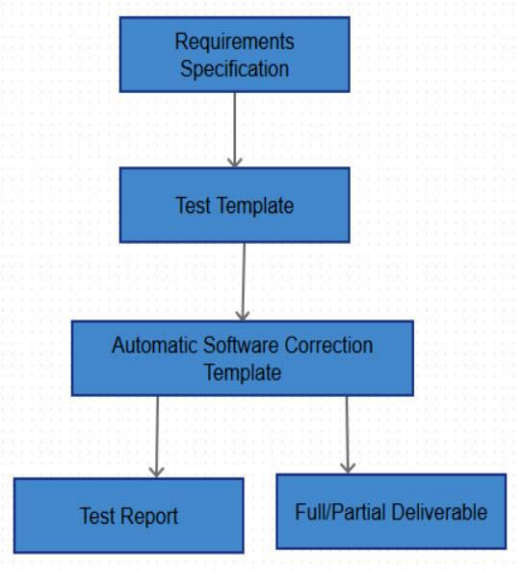

Fig. 6. Automated testing process

2) What is not automated testing?

Software automated testing does not mean translating all manual test cases into a script or test code rather automation is writing tests for best possible scenarios like to provide broader coverage through the tool or software being used. Moreover, test cases that need to be repeated in multiple environments are one of the ideal candidates for automation. While learning the automated testing, we realize manual test cases in a test plan do not have one to one mapping with automated test plans. At times, organizations assume automation as substitute to the manual testing which does not prove to be realistic. A very well-known example is Windows Vista release which went through with lots of inconsistencies making way to the end product and none of them was identified by the automated scripts. Interestingly, the automated scripts concluded the final report with 100% successful execution. Conclusively, most of the client organizations advised their users to stick with Windows XP instead of Vista as prior was relatively more reliable as compared to later.

E. Concept of Virtual Machines

The organizations running business in distributed environment, particularly in agile world, come across the issue of customers demanding versatile operating environments. Vendor organizations have to manage this issue of versatility by developing same product compatible with numerous operating systems that test engineers have to validate accordingly. The use of virtual machines makes it easy to build and test applications on different operating systems. Firstly, agile based software developing organizations break down and manage user stories in backlog management systems. Secondly, they leverage virtualization platform to meet target objectives of producing and testing software systems interoperable with let's say Windows 7, Windows XP, Windows Vista and also all combinations with different OS architectures like x86 and x64 i.e. 32 bit and 64 respectively.

1) Testing on Virtual Machines

From testing perspective, quality assurance teams manage their certification tasks through preparing and running different virtual machines based on respective client' requirements. For example, a particular client demands for a software product running on Windows 7 x64 bit architecture to meet his business needs. The software vendor will develop the system for the said operating environment that test engineers will have to validate on same OS using a virtual machine for Windows 7 x64. In nutshell, the use of virtualization in quality assurance is useful in many perspectives like:

- Software testers can save good amount of time on configuring test platforms.
- When software developers have access to virtual machines demonstrating found or known defects then identification and fixing of bugs becomes easier.
- Virtual machines provide the facility of rollback to any of the previous states if the current state fails or crashes.
- We can create as many numbers of users as required on physical environment and can opt the configuration of our choice while performing testing on a virtual machine.

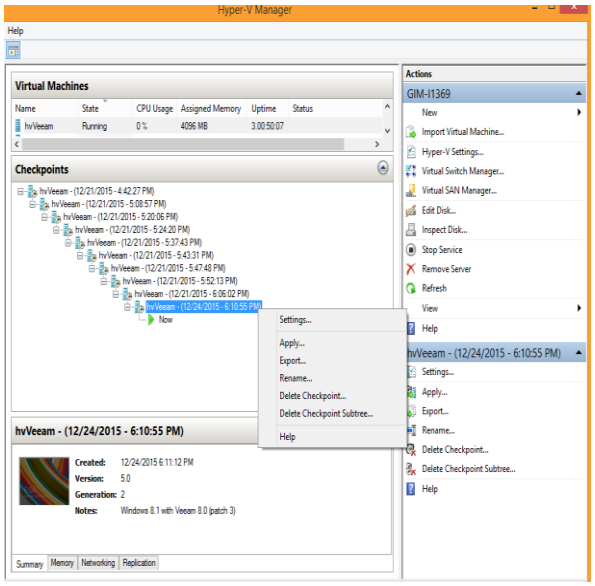

Fig. 7. Checkpoints in virtual machine

2) Checkpoints in Virtual Machines

The support of preserving a particular state of the system in form of snapshot proves to be very useful especially for testers. There are some tools available to manage and work on virtual machines like VMWare Workstation, Hyper-V Manager etc. to name a few. These tool offer the option to create snapshot (in VMWare Workstation) and checkpoint (in Hyper-V Manager) that software programmers or testers create with an intent of preserving the system state in case they have to reproduce a bug or restore to a specific version of product under development or test at a later point. Both of the above mentioned software for virtual machines manage checkpoint in hierarchical format like a tree. Users name individual checkpoints which are customizable. Primarily, checkpoint names are comprehensive representing the OS, system architecture, version of the product installed and date checkpoint created on. Figure 6 below shows the management of checkpoint.

F. Release Readiness Review (RRR) Criteria

In [8] the idea of Release Readiness Review is to certify a combination of checks necessary before rolling out a software release. In agile methodology, a software product is assessed with respect to RRR document at the end of final iteration. The RRR document validates the checklist like: user requirements have been developed and tested; the documentation work has been completed and is available for user; the pending problems pertaining the release have been accommodated; the end product is safe to be run in the client's environment; in case user specific scenarios are required, if any, they are mentioned in known issues section.

The proposed mechanism in Fig. 8 below represents the order of software testing techniques to develop and deliver software products of good quality considering the limited resources under agile methodology.

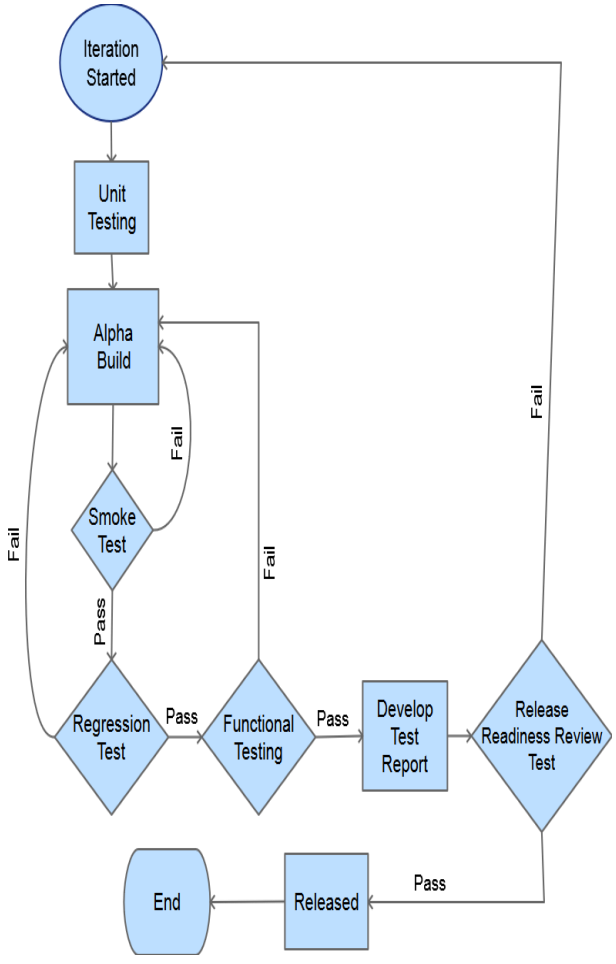

Fig. 8. Proposed order of testing techniques in agile

G. Algorithm

The word Algorithm below has been devised from the above given order of software testing techniques in scrum model.

Stage I: Start of the iteration
Stage II: Execution of Unit Testing on System Under Test (SUT). The outcome of this activity is Automated Test Plans (ATPs).
Stage III: Preparation of Alpha Build to be given to testing team.
Stage IV: Running the Smoke Test Plan
IF (Smoke Test Passed)
i. Publish Smoke Test results
ii. Go to Stage V
ELSE
Go to Stage III
Stage V: Execute Regression Testing
IF (Regression Test Passed)
i. Publish Regression Test results
ii. Go to Stage VI
ELSE
Go to Stage III
Stage VI: Perform Functional Testing
IF (Functional Test Passed)
i. Publish Functional Test results
ii. Go to Stage VII
ELSE
Go to Stage III
Stage VII: Develop Test Report
i. Print test results
ii. Go to Stage VIII
Stage VIII: Assess Release Readiness Review Criteria
IF (RRR Passed)
i. Release the software product
ii. End process
ELSE
Go to Stage I.

IV. Results

There are three basic dimensions derived through the proposed optimized order of testing techniques based on the algorithm developed above: systematic test process in scrum, opportunity for Application Program Interface (API) testing prior to developing alpha build and quick evaluation of build's stability.

A. Systematic Test Process in Agile

The proposed order of software testing techniques provides us a systematic testing process. In scrum methodology, software development process is based on successive iterations where each iteration begins with a sprint planning meeting and ends on a sprint review meeting. From testing aspect, all stakeholders of the product plan and review their work including testing progress. The proposed order analyzes testing progress systematically, leads test team to appropriate stage of the process advising the right testing technique.

B. Opportunity of Application Program Interface Testing

Traditionally, software testing is performed once the end product is built and it comes under the dedicated testing phase of the project. The proposed testing order and algorithm optimize the test process giving an opportunity to test and reveal bugs in the underlying API of the product under development. At times, there are potential logical bugs in the software that remain uncovered and eventually are reported by the customer after releasing the product. We have tried to address this issue in this research work putting the API testing in form of unit testing before making an alpha build available for testing. In agile methodology, unit testing performed on an API generates very useful results finding logical errors that are reported through bug tracking systems like Team Foundation Server (TFS), VersionOne, and Flawtrack which are very effective in scrum based development.

C. Quick Evaluation of Build's Stability

This research contribution recommends performing smoke testing on a software build before any detailed testing taking several hours that brings useful results to know the stability of the build which saves significant amount of testing time. In case smoke test passes, testing process moves to the next stage, otherwise testing order leads to the previous stage

D. System Validity Estimation

Fig. 9 portrays how different testing techniques appear to be effective in the particular order in a series of iterations while working in scrum. In this scenario, 4 iterations have been considered in a project where each iteration lasts for 6 weeks. The unit testing yields significant hours saving in terms of testing effort as it uncovers bugs in the software in very early stage of the project life cycle.

In continuation, functional testing reveals bugs and issues when it comes to testing the functionality of features offered by the product that saves time making developers and testers focus on other critical tasks. In agile development process, the execution of smoke testing and regression testing techniques at appropriate stage of the project offers dual advantages. First, these activities measure health of the product in minimal amount of time. Secondly, they explicitly focus relevant areas of the application under test where changes or bug fixing was made ensuring effort of the team gets put in right dimension.

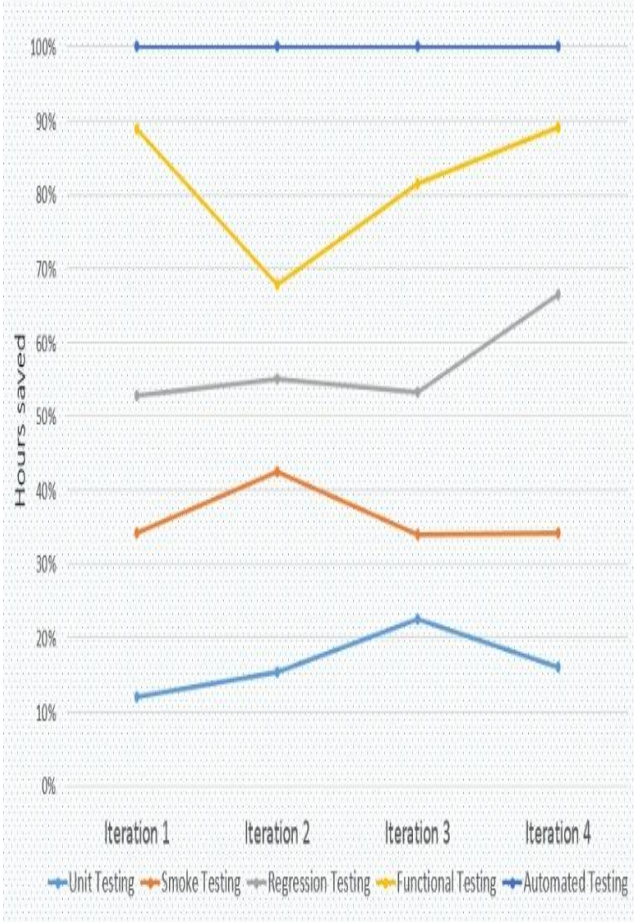

Fig. 9. Validity estimation of optimized order of testing techniques

V. Conclusion and Future Work

The preferred following the agile methodology provides little time cushion to software testing team for exercising testing operations to reveal defects and issues in the product under test that makes software testing a challenge for the test managers. We have presented a combination of software testing techniques in agile that give software testers an opportunity for executing appropriate testing technique at relevant phase while working in scrum. The proposed model takes into account software testing methods like smoke testing, Automated Test Plans (ATPs) in unit testing and regression testing to assess health and stability of an alpha build under testing in a particular sequence. With the execution of aforementioned model, it addresses software testing aspects like manual testing, automated testing and Application Program Interface (API) testing achieving maximum code coverage testifying a broader range of software aspects.

Although, we have devised a testing framework to be considered in scrum model that can provide software testers encouraging feedback regarding adopting appropriate testing approach at a particular stage of software testing process, the future direction could be the complete automation of software testing process. The complete automation may involve automated testing activities ranging from downloading an alpha build, generating test cases automatically, performing the particular testing technique, analyzing test results and generating a comprehensive test report to be shared with the team.

References

[1] J. Singh, "Algorithm and framework for testing and implementation technique in automation of university," no. 2, pp. 140–148, 2016.

[2] M. Ali and T. Saha, "A proposed framework for full automation of software testing process," *Informatics, Electron. Vis. (ICIEV), …*, pp. 436–440, 2012.

[3] J. Berłowski, P. Chruściel, M. Kasprzyk, and I. Konaniec, "Highly Automated Agile Testing Process : An Industrial Case Study," vol. 10, no. 1, pp. 69–87, 2016.

[4] V. K. Chauhan, "Smoke Testing," *Int. J. Sci. Res. Publ.*, vol. 4, no. 1, pp. 2250–3153, 2014.

[5] D. S. Cruzes, N. B. Moe, and T. Dybå, "Communication between Developers and Testers in Distributed Continuous Agile Testing," 2016.

[6] N. Frechette, L. Badri, and M. Badri, "Regression Test Reduction for Object-Oriented Software: A Control Call Graph Based Technique and Associated Tool," *ISRN Softw. Eng.*, vol. 2013, no. 2013, pp. 1–10, 2013.

[7] G. Di Fatta, "KNIME as a Teaching Tool in Higher Education," vol. 01107, 2013.

[8] D. Port and J. Wilf, "The value of certifying software release readiness: An exploratory study of certification for a critical system at JPL," *Int. Symp. Empir. Softw. Eng. Meas.*, pp. 373–382, 2013.

[9] S. Shamshiri, R. Just, J. M. Rojas, G. Fraser, P. McMinn, and A. Arcuri, "Do automatically generated unit tests find real faults? An empirical study of effectiveness and challenges," *Proc. - 2015 30th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2015*, pp. 201–211, 2016.

[10] A. Brooks, J. Chambers, C. N. Lee, and F. Mead, "A partial replication with a sample size of one: A smoke test for empirical software engineering," *Proc. - 2013 3rd Int. Work. Replication Empir. Softw. Eng. Res. RESER 2013*, pp. 56–65, 2013.

[11] P. Singh and P. Patel, "Impact of agile testing over traditional testing," vol. 1, no. 2, 2015.

[12] S. M. Shahabuddin and Y. Prasanth, "Integration testing prior to unit testing: A paradigm shift in object oriented software testing of agile software engineering," *Indian J. Sci. Technol.*, vol. 9, no. 20, 2016.

[13] K. Schwede and K. Tucker, "A survey of test ideals," vol. 105, no. 4, p. 44, 2011.

[14] Y. Shin and H. Mark, "Regression testing minimization, selection and prioritization: a survey," *Softw. Testing, Verif. Reliab.*, pp. 67–120, 2010.

[15] V. Garousi and N. Koochakzadeh, "Testing – Practice and Research Techniques," *Test. - Pract. Res. Tech. Proc. 5th Int. Acad. Ind. Conf. TAIC PART 2010*, vol. 6303, no. October 2016, 2010.

[16] C. J. Hunt, G. Brown, and G. Fraser, "Automatic testing of natural user interfaces," *Proc. - IEEE 7th Int. Conf. Softw. Testing, Verif. Validation, ICST 2014*, pp. 123–132, 2014.

[17] C. T. Lin, K. W. Tang, C. D. Chen, and G. M. Kapfhammer, "Reducing the cost of regression testing by identifying irreplaceable test cases," *Proc. - 2012 6th Int. Conf. Genet. Evol. Comput. ICGEC 2012*, pp. 257–260, 2012.