International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Benefits of Test Automation for Agile Testing

Manu GV[1], Namratha M[2], Pradeep[3]

[1]Technical Lead-Testing Calsoft Labs, Bangalore, India
[2]Assistant Professor, BMSCE, Bangalore, India
[3]Software Engineer, Cerner, Bangalore, India

Abstract: ***Software testing is essential and most important for any application to work correctly under customer requirements. Proper testing techniques must be adopted to suit the application and so as to enhance the performance and ease the maintain ability. In this paper, we present agile methodology of testing which is the most popular now since it takes into consideration the changing customer needs. A comparison of agile testing with V model is also presented. Automation testing using the Open source tool Selenium is compared with manual testing techniques which were used earlier. Test Automation today is developing as a separate discipline and organizations are now thinking towards automating application, to have a better cost benefit in the long run. But there are many challenges which an organization has to face when they want to automate their application.***

Keywords: Agile testing, Automation, Continuous integration, Customer feedback, Selenium, V-model

1. Introduction

Software testing is the process of evaluation of a soft-ware item to detect differences between given input and expected output and also to assess the feature of a software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process for better quality software. In other words software testing is a verification and validation process.

Verification is the process to make sure the product sat-isfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to. Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

If software testing is carried out completely on the early stages in the product life cycle then software production will be more inexpensive in the long run period. Software bugs can be found at any moment of time in the life cycle, from such early stages as conversations about the potential architecture to such late stages as a client calling the support desk regarding an error he has met. As time passes and the life cycle of product progresses it becomes increasingly more expensive to fix software bugs.

2. Agile Testing

Agile as the name refers implies something to do very quickly. Hence Agile Testing [1] refers to validate the client requirements as soon as possible and make it customer friendly. As soon as the build is out, testing is expected to get started and report the bugs quickly if any found. As a Tester, we need to provide our thoughts on the client requirements rather than just being the audience at the other end. Emphasis has to be laid down on the quality of the deliverable in spite of short timeframe which will further help in reducing the cost of development and your feedbacks will be implemented in the code which will avoid the defects coming from the end user. In the modern economy, it is often difficult or impossible to predict how a computer-based system e.g., a web-based application will evolve as time passes. Market conditions change rapidly, end-user needs evolve, and new competitive threats emerge without warning. In many situations, you won't be able to define requirements fully before the project begins. You must be agile enough to respond to a fluid business environment. Fluidity implies change, and change is expensive. Particularly if it is uncontrolled or poorly managed, One of the most compelling characteristics of the agile approach is its ability to reduce the costs of change throughout the software process. Does this mean that recognition of challenges posed by modern realities causes to discard valuable software engineering principles, concepts, methods, and tools? Absolutely not, like all engineering disciplines, software engineering continues to evolve. It can be adapted easily to meet the challenges posed by a demand for agility.

2.1 Benefits of Agile Testing

1. Development and testing activities are concurrent.
2. Everyone work as a team towards a common goal and everyone is responsible for the quality.
3. Continuous integration [2] and customer feedback.
4. Less risk of squeezed time period.
5. Working software is developed and delivered to the customer frequently and documented.

2.2 Challenges of Agile Testing

1. Inadequate Test Coverage - With continuous integration and changing requirements, it can be easy to miss critical tests for any requirement. This can be mitigated by linking tests to user stories for better insight into test coverage and analyzing specific metrics to identify traceability and missing test coverage. Another cause of missing test cover-age is due to code being changed that was not anticipated. To mitigate that source code analysis is needed to identify modules that were changed to ensure that all changed code is properly tested.

2. Code Broken Accidentally due to Frequent Builds - Since code is changed and compiled daily, the likelihood of code breaking existing features is much higher. To attack this issue you must have a way of running a series of tests against each build. Since most of us are resource con-strained, it is not practical to have testers do this daily so we must rely on automated testing to do this for us.
3. Early Detection of Defects [3] - Defects are substantially more expensive to fix later in the development cycle. In other words, if you find a defect during requirements definition, it is much cheaper to fix and has less impact on future coding than those found late in the testing cycle or even worse, in production. To resolve this issue, your team can do frequent code reviews to spot issues early. Another option is to run static analysis tools on your source code -- these are great at finding missing error routines, coding standard derivations, and data type mismatch errors that can crop up in production.

Inadequate API Testing - Most software is now designed with a service orientated architecture that exposes their APIs publicly so that other developers can extend the solution. For those of us developing APIs, it can be easy to overlook API testing because of the complexity of doing it. Many testers do not have the skills to test APIs because it normally requires strong coding skills to do so. To prevent missing API tests, there are tools that allow testers to test the API without strong coding skills, so this is a great way to ensure that these services are fully tested.

Performance Bottlenecks - As software becomes more mature, complexity normally increases. This complexity adds more lines of code which introduces performance issues if the developer is not focused on how their changes are impacting end-user performance. To solve this issue, you must first know what areas of your code are causing performance issues and how performance is being impacted over time. Load testing tools can help identify slow areas and can track performance over time to more objectively document performance from release to release.

3. Strategies of Agile Testing

1. Early Software Testing:

- We want to test as early as possible because the potential impact of a defect rises exponentially over time (this isn't always true, but it's something to be concerned about). In fact, many agile developers prefer a test-first approach.
- We want to test as often as possible, and more importantly, as effectively as possible, to increase the chance that you'll find defects. Although this increases your costs in the short term, studies have shown that greater investment in testing reduces the total cost of ownership of a system due to improved quality.
- We want to do just enough testing for our situation such as Commercial banking software requires a greater in-vestment in testing.
- Pair testing [4], just like pair programming and modeling with others, is an exceptionally good idea.

2. Testing Throughout the Lifecycle:

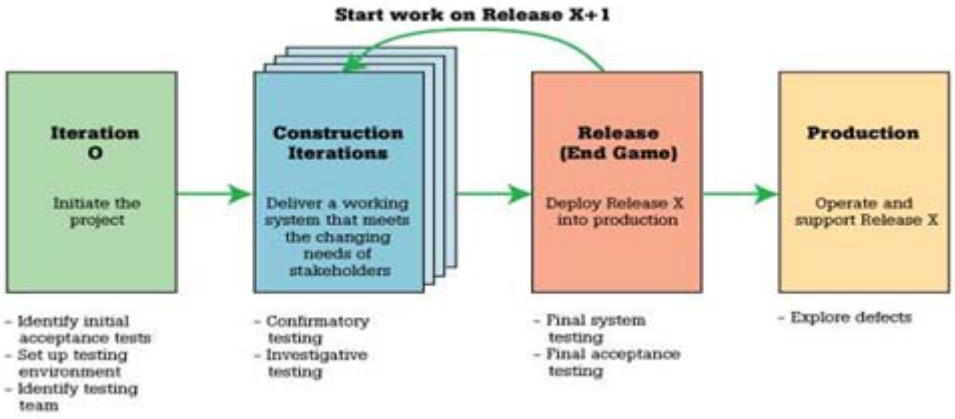


Figure 1: Test activities during the agile lifecycle

Testing activities vary throughout the lifecycle. During Iteration 0, we perform initial setup tasks. This includes identifying the people who will be on the external testing team, identifying and potentially installing testing tools. If the project has a deadline, we would want to identify the date into which our project must enter the End Game.

A significant amount of testing occurs during construction iterations—agilists test often, test early, and usually test first. This is confirmatory testing against the stakeholder's current intent and is typically milestone-based at the unit level. This is a great start, but it's not the entire testing picture. Regardless of the style, our true goal should be to test, not to plan to test, and certainly not to write comprehensive documentation about how you intend to hopefully test at some point. Agilists still do planning, and we still write documentation, but our focus is on high-value activities such as actual testing.

During the End Game, we may be required to per-form final testing efforts for the release, including full sys-tem and acceptance testing. The testing effort is greatly reduced at the End Game since rigorous testing is done at initial stages only.

3. Testing During Construction Iteration:

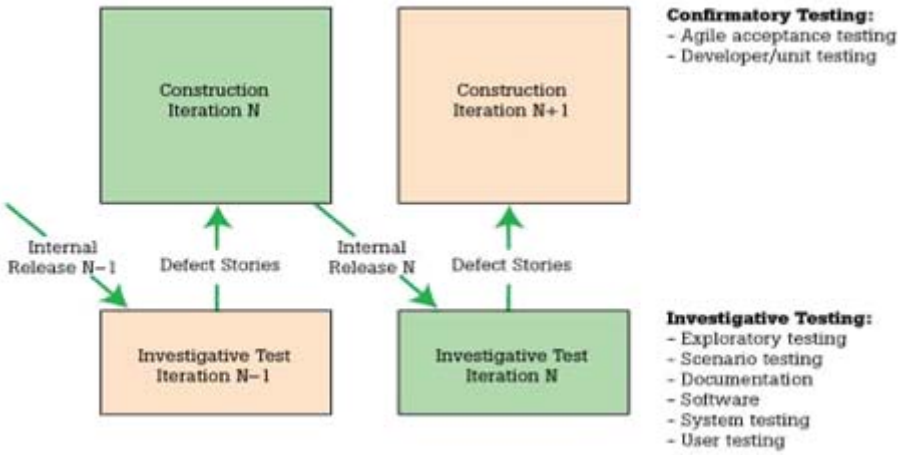


Figure 2: Incremental testing throughout the agile development lifecycle.

There are two aspects to confirmatory testing: agile acceptance testing and developer testing, both of which are automated to enable continuous regression testing throughout the lifecycle. Confirmatory testing is the agile equivalent of testing to the specification, and we consider acceptance tests to be the primary part of the requirements specification and our developer tests to be the primary part of the design specification. Both of these concepts are applications of the agile practice of single sourcing information whenever possible.

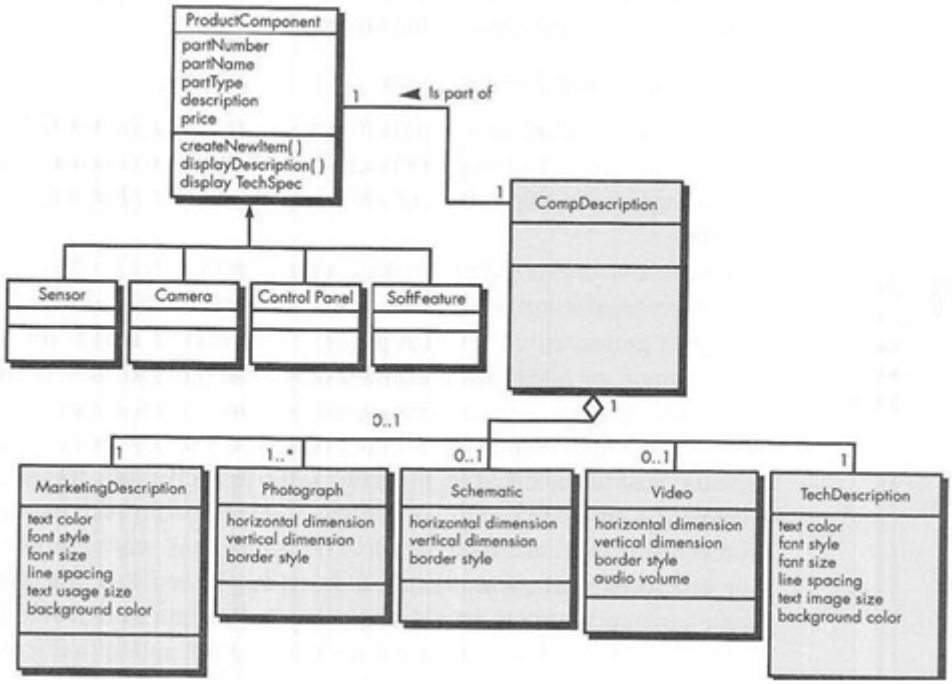
Agile acceptance testing is a mix of traditional functional testing and traditional acceptance testing because the development team and their stakeholders are doing it collaboratively. Developer testing is a mix of traditional unit testing and traditional class integration testing. Our goal is to look for coding errors, perform at least coverage if not full path testing, and to ensure that the system meets the cur-rent intent of its stakeholders. Developer testing is often done in a test-first manner, where a single test is written and then sufficient production code is written to fulfill that test. This test-first approach is considered a detailed design activity first and a testing activity second.

Automation is an important aspect of construction testing due to the increased need for regression testing on evolutionary projects. It is possible to generate acceptance test cases from use cases and scenario definitions or from process diagrams such as UML activity diagrams or flow

4. Investigative Testing:

Investigative testers describe potential problems in the form of defect stories—the agile equivalent of a defect report. A defect story is treated as a form of requirement—it is estimated and prioritized and put on your requirements stack. The need to fix a defect is a type of requirement, so it makes perfect sense to address it just like any other requirement. As we would expect, during the End Game the only requirement type that we're working on is defect stories. Good investigative testing efforts reveal any problems [6] that developers missed long before they become too expensive to address. It also provides feedback to management that the team is successfully delivering high-quality working software on a regular basis.

Let us consider a particular scenario of purchasing a product which can be schematically represented using UML notation as shown below. Let us now see how agile testing is useful in this case.



4. Using Selenium for Automation Testing

The main challenges associated with automation testing are:

1. Selection of Automation Tool – Today there are n number of automation tools available in the market and choosing a good automation tool is one of the major challenges that an organization often faces. This is majorly because there are several commercial tools which are expensive, there are open source tools which might not be reliable and there are tools of which an organization may not have sufficient expertise to make optimum use of.
2. No Defined Process for Executing Automation Project within an organization – Automation is like a project execution, wherein you start with Requirement, then you design the framework as per your requirements and finally you roll it out for Test Case Execution. Lack of systematic approach and process will make successful automation really tough. As we have processes, guidelines and checklist defined for our development project; we should also have guidelines, processes and checklist available for Automation as well.
3. Availability of Right Resources – The right set of resources is a must when you are doing automation. This means the resources should be skilled enough to design and code robust scripting, so that it requires minimum time for debugging during maintenance.
4. Commitment from Customer or Management – Automation is time consuming and resource intensive task. Customer or management commitment is required if one wants to get the real benefit of automation. This paper will highlight the approach wherein you can maximize your benefits within reasonable period of time.

Using selenium to overcome these challenges:

Selenium [10] is an Open Source Tool developed by Thought Works. There are a set of Selenium tools which when combined provides you the power to automate simple to complex web application. Some of the Selenium Tools under ThoughtWorks umbrella are:

- Selenium IDE – It is a Firefox add-on that makes it easy to record and playback tests in Firefox 3+. You can even use it to generate code to run the tests with Selenium Remote Control.
- Selenium Remote Control (RC) – It is a client/server system that allows you to control web browsers locally or on other computers, using almost any programming language and testing framework.
- Selenium Grid – It takes Selenium Remote Control to another level by running tests on many servers at the same time and cutting down on the time it takes to test multiple browsers or operating systems.

Selenium is the best open source tool for doing automation for web based application and it does not have any cost attached to it. The only cost is the effort which will go for designing and developing the script for the application. There is no need to define or develop separate Life Cycle for doing automation in Selenium. If you have an existing automation process, Selenium Automation Life Cycle will fit into that well. If you are using Selenium as an automation tool, you will find a lot of information online for the best process or Life Cycle to be adopted for automation. Selenium scripting can be done in a number of programming languages like C#, Java, PHP, Ruby etc, unlike other commercial tools which support single scripting language. Since Selenium scripting can be done in any language of choice, one can easily find right resources for the programming language chosen for Selenium Automation. Last but not the least, since this tool comes at ZERO PRICE, an organization's

management will find that the only investment they have made is on the infrastructure and human effort and not on the heavy licensing cost.

Application Name	*XYZ*
Number of Test Cases	4000 (Sample Number)
Manual(hours) to execute	336 (Considering 5 Min is required for one Test Case Execution)
Automated Effort (hours)to execute	132 (Considering 2 Min for one Test Script)
Number of Test Iteration Planned Yearly	20 (20 Iteration are planned in a Year. Each iteration will require execution of 2000 Test Cases)
Total Projected Hours Saved	(336*10) – (132*20) = 720
Total Savings Annually ($)	720 * 50 =36,000 (Considering 50$ per hour is one FTE Rate)
Savings %	(720/3360)*100 = 21.4%
Automation Effort Estimate	4000 (Considering 2 hours for Scripting one Test Case)
Automated Effort FTE Cost ($)	4000 * 60 = 2,40,000 (Considering 60$ per hour is one FTE Rate)

References

[1] http://www.testingexperience.com/testingexperience03_09.pdf
[2] http://agile.csc.ncsu.edu/SEMaterials/AgileTesting.pdf
[3] http://www.mattivuori.net/julkaisuluettelo/liitteet/agile_testing.pdf
[4] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6298092&contentType=Conference+Publications&sortType%3Dasc_p_Sequence%26filter%3DANDD(p_IS_Number%3A6298076)
[5] http://132.68.98.62/Courses/cs_methods/eXtremeProgramming/XP_Papers/Testin g_IEEE_Software.pdf
[6] https://sw.thecsiac.com/databases/url/key/2399/8462
[7] http://www.google.co.in/url?sa=t&rct=j&q=v+model&source=web&cd=7&cad=rja&ved=0CEAQFjAG&url=http%3A%2F%2Fwww.onestoptesting.com%2Fsdlc-models%2Fv-model.asp&ei=0khsUKfqD8zirAe7p4CIDg&usg=AFQjCNH1Rcpj2sBmaAl0AVM-0zROFKK-ew
[8] http://www.ijcaonline.org/journal/number12/pxc387425.pdf
[9] http://seleniumhq.org/docs/book/Selenium_Documentation.pdf
[10] http://www.google.co.in/url?sa=t&rct=j&q=v+model+in+software+testing&source=web&cd=10&cad=rja&ved=0CFUQFjAJ&url=http%3A%2F%2Fwww.softwaretestingclass.com%2Fv-model%2F&ei=30hsUM2PBY-zrAfFwoCoCg&usg=AFQjCNFN5TBPRyMuIk0BwtUPaQTPJeYRXA