www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 5 Issue 09 September 2016 Page No.17996-18000

Agile Testing Automation Framework

Heet Palod[1], Danish Chaus[2]
[1]Department of Computer Engineering, Fr. Conceicao Rodrigues Institute of Technology,
Sector 9A, Vashi, Navi Mumbai, India
heetpalod13@gmail.com

[2]Department of Computer Engineering, Fr. Conceicao Rodrigues Institute of Technology,
Sector 9A, Vashi, Navi Mumbai, India
danishkchaus@gmail.com

Abstract: Software testing is one of the most important phases of Software Development Life Cycle and main technique to find bugs and ensure the quality of the software. Software Testing can be conducted manually as well as automated. In manual testing, testing is done without any tool. In automation testing, testing is done with the help of automated testing tools. In recent times, agile development and testing are growing in popularity. Agile testing tools vary from project management tools to automated testing tools. In this paper, we have explained the detailed working of Agile Testing Framework (ATF). The ATF is about implementing the ideal automated testing environment in an agile organization, to deliver quality software, fast. The ATF Architecture can be used to easily integrate with continuous integration tools and best in class third-party testing tools to allow for accessibility, security, web consistency and full performance (front-end, middleware and load) testing.

Keywords: Agile, Test-Driven Development (TDD), Acceptance Test-Driven Development (ATDD), Behavior-Driven Development (BDD), Technical Debt, Regression, Refactor, and Continuous Integration.

1. Introduction

Agile testing differs from the big bang, test-at-the-end approach used in traditional development. Instead, a code is developed and tested in small increments, often with the development of the test itself preceding the development of the code. In this way, tests serve to elaborate and better define the intended system behavior before the system is coded. Quality is built in from the beginning. This just-in-time approach to the elaboration of the intended system behavior also mitigates the need for lengthy, detailed requirements and specifications, and sign-offs, as are often used in traditional software development to control quality. Also these tests, unlike traditional requirements, are automated wherever possible. Even when they're not, they provide a definitive statement of what the system actually does, rather than a statement of early thoughts about what it was supposed to do.

Agile testing is a continuous process, not a one-time or end game event. It is integral to lean and built-in quality. Simply, Agile teams can't go fast without endemic quality, and they can't achieve endemic quality without continuous testing and, wherever possible, testing should be practiced first. [7]

2. Traditional Testing Automation V/S Agile Testing Automation Framework

Traditional, Record-and-Playback, Heavyweight, Commercial test automation solutions are not agile as the test-last workflow encouraged by such tools is inappropriate for agile teams and also unmaintainable scripts are created with such tools which are an impediment to change. The practice of these tools forces teams to wait until after the software is done or at least the interface is done before automation can begin. After all, it's hard to record scripts against an interface that doesn't exist yet. So the usual workflow for automating tests with a traditional test automation tool follows these phases:

- Test analysts design and document the tests
- Test executors execute the tests and report the bugs
- Developers fix the bugs
- Test executors re-execute the tests and verify the fixes (repeating as needed)
- Test automation specialists automate the regression tests using the test documents as specifications.

By the time it gets around to automating the tests, the software is ready so tests are not going to uncover much information that we don't already know. Automated regression tests are theoretically handy for the next release but usually, the changes made for the next release breaks the automation. The result for most contexts is high cost and limited benefit. However, this workflow is particularly bad in an agile context where it results in an intolerably high level of waste and too much feedback latency.

- Waste: The same information is duplicated in both the manual and automated regression tests
- Feedback Latency: The bulk of the testing in this workflow is manual and it takes days or weeks to discover the effect of a given change.

Traditional test automation tools don't work for an Agile context because they solve traditional problems, and those are different from the challenges faced by Agile teams. Agile teams need fast feedback that automated system/acceptance tests provide. Further, test-last tools cannot support Acceptance Test

Driven Development (ATDD). Agile teams need tools that support starting the test automation effort immediately, using a test-first approach. [8]

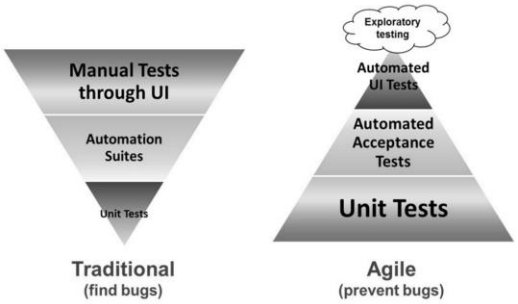

Fig. 1: Traditional V/s Agile Testing Automation

3. Characteristics of Agile Testing Automation Tools

To tackle the problems faced in traditional test automation tools, Agile teams need test automation tools/frameworks that:

- Support starting the test automation effort immediately, using a test-first approach.
- Separate the essence of the test from the implementation details.
- Support and encourage good programming practices for the code portion of the test automation.
- Support writing test automation code using real languages, with real IDEs.
- Foster collaboration.

Agile teams don't need tools optimized for non-programmers. Agile teams need tools to solve an entirely different set of challenges related to collaborating, communicating, reducing waste, and increasing the speed of feedback. There must be a balance in the quality goals of the team with the requirements and risk of individual user stories. [8]

4. Deciding, Maintaining & Reviewing Automated Tests in Agile Environment

Developers and testers working on the story make detailed decisions about where automation can be used to trigger acceptance/signal completion of the story. During testing, testers may see opportunities for ad-hoc, on the fly automation to increase test coverage.

Since in Agile testing, tester-developer and/or tester-product owner pairs often create tests, there's a built-in review process of the test scope and validation level as part of story acceptance. Otherwise, the collaborative planning to define the tests, then the collaborative acceptance with the product owner provides ample shared knowledge and review. The end result is robust, resilient automated tests, which provide the backbone of regression testing, and any test failures from regression testing may also trigger a test review. Ideally, the entire team maintains all tests, whether unit, functional, or any other. However, testers may use specialized tools for GUI-level tests that require their involvement to maintain tests over time. To reduce the possible soloing of this information, the tools should be carefully chosen to be accessible to all team members, information should be readily available, and the test inventory should be continuously culled to prevent test bloat.

For functional/acceptance tests, it's most common for there to be a combination of white-box (under-the-covers) and UI automation (black-box) tests. The white-box tests tend to be less fragile, so are less costly to maintain, whereas the black-box UI-driven tests need to be leveraged judiciously due to the cost to develop and maintain the tests over time. Ideally, testers are involved in both, so that the black-box tests can supplement instead of duplicating white-box tests. [7]

5. Feature of Agile Testing Automation Framework

Most agile teams, regardless of any specific methodology, are looking for fast, reliable feedback from their automated tests. Fast, reliable feedback means timely results with minimal false failures, ideally incorporated as part of the continuous integration (build) pipeline. This makes Agile Testing framework more suitable for customer relationship development-oriented.

6. Working of Agile Testing Automation Framework

Agile Testing Framework consists of the following:

6.1 Agile Development Process:

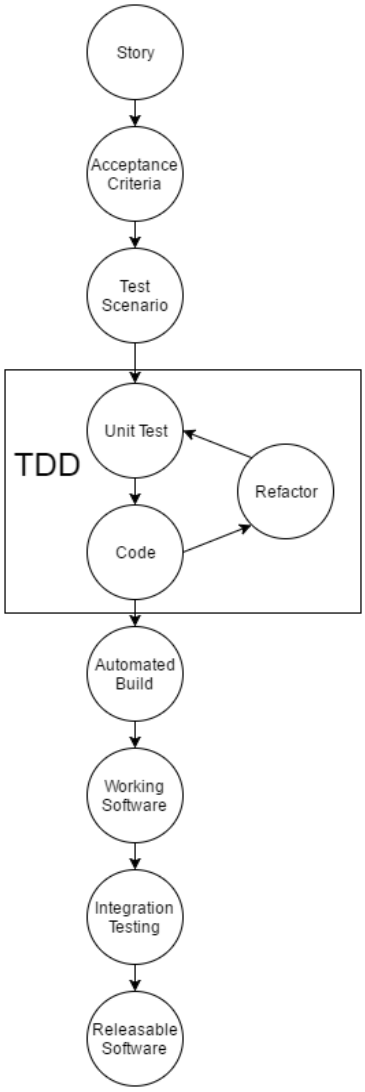

Fig. 2: Agile Development Process Framework

6.1.1 Stories:

Stories provide the features/capabilities required by the user. This phase describes what is needed to test so that the user will be able to complete the action.

6.1.2 Acceptance Criteria:
These cover the boundaries of the story and will serve as scenarios for writing the test cases. Once the story is written or during story writing, it's important to capture acceptance criterion and make sure that an acceptance criteria is not actually a story or vice versa. The acceptance criteria do not expand the story, it strictly provides the scenario behind a story, in order to write good test cases.

6.1.3 Test Scenarios:
Scenario testing is a software testing activity that uses scenarios: hypothetical stories to help the tester work through a complex problem or test system. The ideal scenario test is a credible, complex, compelling or motivating story the outcome of which is easy to evaluate.

6.1.4 BDD – Behavior Driven Development:
The test suite acts as a regression safety net on bugs: if a bug is found, the developer should create a test to reveal the bug and then modify the production code so that the bug goes away and all other tests still pass. On each successive test run, all previous bug fixes are verified. It also reduces debugging time.

BDD/ATDD provide a way to help build confidence in the automated tests, first by expressing all requirements in high-level business terms and then by automating these requirements in a way that provides a set of living/executable documentation detailing both which requirements were requested and how they have been implemented. BDD provides both a single source of truth about the application's behavior and also a set of regression tests protecting it against unwanted change.

6.1.5 TDD - Test Driven Development:
Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards. Some of the major advantages of TDD are code coverage; regression testing, simplified debugging and the test cases can be used for system documentation. [3]

6.1.5.1 Phases of TDD:

- Unit Test:
  A unit test verifies that a function or set of functions "meets the acceptance criteria" – in other words, that the function(s) under test meet the requirements. Unit Tests can be written using TestNG or jUnit for java applications and using MSTest or nUnit for applications developed using .NET programming language.
- Code:
  In TDD, test cases are written before the implementation of the code. The developer develops the code based on the test cases.
- Refactor:
  Now the code should be cleaned up as necessary. Move code from where it was convenient for passing the test to where it logically belongs. Remove any duplication can find. Make sure that variable and method names represent their current use. Clarify any constructs that might be misinterpreted. [3]

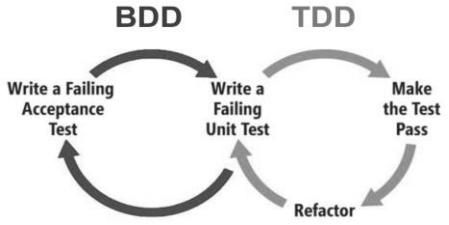

Fig. 3: Behavioral-Driven Development (BDD) & Test-Driven Development (TDD)

6.1.6 ATDD - Acceptance Test-Driven Development:
Acceptance tests are from the user's point of view – the external view of the system. They examine externally visible effects, such as specifying the correct output of a system given a particular input. In general, they are implementation independent, although automation of them may not be. Acceptance tests are a part of an overall testing strategy. Acceptance tests are created when the requirements are analysed and prior to coding. Failing tests provide quick feedback that the requirements are not being met. The tests are specified in business domain terms. The terms then form a ubiquitous language that is shared between the customers, developers, and testers. Tests and requirements are interrelated.

A requirement that lacks a test may not be implemented properly and a test that does not refer to a requirement is an unneeded test. An acceptance test developed after implementation begins represents a new requirement. Acceptance criteria is a description of what would be checked by a test. [1]

6.1.7 Automated Build:
Many organizations make the mistake of testing their code possibly on a daily but more likely on a weekly basis, this leaves too much room for defects to creep in while we're not looking. By the time a bug is discovered, more code has been layered on top of it – making it harder and more expensive to fix. Testing changes right away dramatically reduces the cost of addressing defects, so testers should kick off a build with each commit, or at the very least on scheduled intervals throughout the day.

6.1.8 Working Software:
Mentioned as one of the four values from the Agile Manifesto, "Working software over Comprehensive documentation" and also in several of the principles, we must recognize the importance of producing working software, within an iteration. This means that not only the software respects agreed upon testing standards, but also has passed all the tests included in the Definition of Done for the organization. In Scrum, it's referred to as "potentially shippable product increment", which means that if the work as been accepted then it could ship immediately, and therefore must have been thoroughly tested.

6.1.9 Integration Testing:
Integration testing is usually the earliest actual testing that can find integration bugs (although review processes can find some bugs before the software is written, or before the code goes to test). One would find these bugs before showing the software to the product owner or releasing it since fixing bugs at the last moment is very expensive. One would find these bugs earlier in

the process when they would be cheaper to fix because one might need multiple working components to integrate. Integration testing ensures that the units are working together in concert. This is mainly to test our design. If something breaks here, we have to adjust our unit tests to make sure it doesn't happen again.

6.1.10 Releasable Software:
Once a feature is complete and all the testing has been passed for that feature, then it is deemed Releasable Software.

6.2 Continuous Delivery:

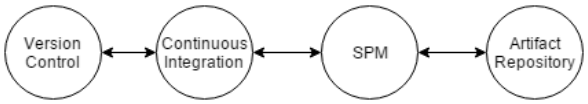

Fig. 4: Continuous Delivery Process in Agile Testing Automation Framework

6.2.1 Version Control:
Source control is a must for modern collaborative software development. There are many different source control tools and solutions available, they range from commercially licensed (such as Team Foundation Server (TFS) or ClearCase) to open source ones (such as GIT, SVN, or Mercurial). If a code is in source control, it is versioned, it is available to anyone who has access, and it is secure.

6.2.2 Continuous Integration:
Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. An automated build, allowing teams to detect problems early, then verifies each check-in. [5]

6.2.3 Software Project Management:
Software project management tools such as Maven are primarily used for building Java projects. Software project management tools addresses two aspects of building software: First, it describes how software is built, and second, it describes its dependencies.

6.2.4 Artifact Repository:
A key part of DevOps is building a delivery pipeline, which is capable of deploying development versions of our binary artifacts to a component repository and to a target environment (e.g., deploying a web application to a web container).

6.3 Reporting:

With the industry-wide movement towards DevOps, the need for organizations to be able to rapidly adjust, make strategic decisions or change the direction of their software development based on rapid feedback (Feedback Loops) and metrics/measurement. The problems many organizations run into are that they have no feedback loop or metrics/measurement in place.

DevOps brings the level of metrics and measurement to a completely different level through its extensive use of monitoring tooling. [6]
Some of the DevOps Monitoring Tools are:

- Graphite
- Tasseo
- SonarQube
- Ganglia
- Nagios
- Collectd
- Munin
- Pingdom
- New Relic
- DATADOG

6.3.1 Technical Debt:
Technical debt is a concept in programming that reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution. [4]

Symptoms of Technical Debt:

- Loss of Productivity
- Increase in Testing
- Postponed Releases
- Code Duplication
- Low Code Coverage
- Increase in Bugs
- Unreadable Code
- Decreased Velocity
- Using Old Libraries
- Heavy Stress on Approaching Deadlines
- Being scared of Changing Anything
- Wrong Design or choice of technology

How Technical debts can be prevented in Agile Engineering Practices:

- Pair Programming, TDD
- Continuous Integration
- Automated Unit Tests
- Automated Functional Tests
- Automated Other Tests (Regression)
- Refactoring

6.4 Testable Components:
Testable components include:

- Web apps
- Mobile apps
- Databases
- Services
- Reports and Forms

All these components are tested during each iteration cycle.

6.5 Testing Tools:
The different types of testing carried out are:

- *Performance Testing:*
  Performance Testing consists of two parts: Front-end Performance and Middleware performance testing. Front-end performance testing deals with how fast the page loads while Middleware performance testing provides information about the resource consumption of the application using Selenium.
- *Accessibility:*
  Web accessibility solution involves capturing the HTML of each unique page in application visited via WebDriver, then firing the Wave accessibility tool. The DOM is then checked to ensure there are no WAVE errors and

screenshot it and fail the test if there are. This is all done during a dedicated accessibility stage of build pipeline.

- *Security Testing:*
  Combine Selenium and OWASP's Selenium project or Zed Attack Proxy (ZAP) in order to perform easy to use integrated penetration testing for finding vulnerabilities in web applications.
- *Load Testing:*
  The combination of JMeter and BlazeMeter can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types.
- *Web Consistency:*
  Web Consistency Testing is all about automating how users visually interact with the site.

7. Advantages of Agile Testing Automation Framework

The benefits that we experience using automation for agile tests are:

A. *Speed* – Takes less time than the Traditional testing approach.
B. *Quality* – Delivers excellent quality product software without any bugs or defects.
C. *Malleable code base* – Easily adaptable and open to changes.
D. *Rapid feedback* – Provides rapid feedback in order to make necessary changes as and when needed.
E. *Efficiency* – Delivers efficient product software which runs smoothly.

8. Challenges for Implementing Agile Testing Automation Framework

Availability of people & skills is a holistic team-oriented approach, striking the right balance between automation and excess test inventory can result in drag. Some teams struggle with when to automate functional tests – whether to delay acceptance of a user story until the automated tests are complete or to have automated test development as a separate activity. We've found that with the right tools, development of GUI-level tests can easily take place as standard work on a user story. However, automating functional tests on a story-by-story basis may lead to test bloat and may not provide the higher-level coverage that's desired from these tests. Teams may choose to write user stories to prioritize and schedule test development at that level.

9. Conclusion

The main purpose of this paper is to elucidate Agile Testing, one of the latest trends, which have gained significant momentum in the field of testing. It briefly explains how Agile testing collaborates with continuous integration, thus automating its process and providing rapid feedback without any compromise with the quality. It exhibits the importance of test-driven development and explains how automating test cases play a vital role in the development cycle of the system. Most agile teams, regardless of any specific methodology, are looking for fast, efficient and reliable feedback from their automated tests. Agile Testing Automation Framework makes uses of these quality factors and provides timely results with minimal false failures, ideally incorporated as part of the continuous integration pipeline. This makes Agile Testing framework more suitable for customer relationship development-oriented.

References

[1] Crispin, Lisa, and Janet Gregory. Agile Testing:A Practical Guide for Testers and Agile Teams, Addision-Wesley, 2009.
[2] http://scaledagileframework.com/test-first/
[3] http://www.agiletestingframework.com/atf/testing/test-driven-development-tdd/
[4] http://www.agiletestingframework.com/atf/reporting/technical-debt/
[5] http://www.agiletestingframework.com/atf/implementation/continuous-integration/
[6] http://www.agiletestingframework.com/atf/reporting/
[7] https://smartbear.com/learn/automated-testing/testing-in-agile-environments/
[8] http://testobsessed.com/2008/04/agile-friendly-test-automation-toolsframework