

Acceptance Test Driven Development

By

Vishal Aggarwal, Manpreet Singh
Infosys Limited, Chandigarh, India – 161001
Infosys Limited, Chandigarh, India – 161001

vishalaggarwal27@gmail.com, ksinghmanpreet@gmail.com

ABSTRACT:

This document addresses the benefit of acceptance test driven development processes and solutions in the IT domain. Over a past decade it seems that the business requirements are fluidic in nature. In order to ensure quality IT deliverable with minimal rework and lower defect density, in midst of the ever changing business requirement, it's important to ensure following:

- Requirement understanding at granular level since the day of inception.
- Tangible, measurable and testable code development meeting user's requirements.
- Regular feedback based interim automated testing during the course of development cycle.

Keywords:

Agile, Acceptance Criteria, Business Requirement, Acceptance Test Driven Development.

1. INTRODUCTION

One of the major challenges witnessed by product and solution delivering organization are the changing requirements from business owners.

The timing of the change can be anytime during the course of the deliverable like even during the mid-phase of development cycle. Gaps and Defect density tend to increase under such scenarios due to demand been in upswing and aggressive time lines to be met.

Agile unlike the waterfall model is one of the most promising way of executing the projects while supporting the dynamics of getting the regular feedback from the users on the partial

development during walkthrough of the developed code, before taking it on to live.

In midst of these dynamics there is a need of sustainable framework to minimize the impact of changing requirements and thus the wastage of the efforts laid to build up the codebase.

Automated test suites that can be integrated with continuous integration servers in this direction becomes the need of an hour.

2. TEST DRIVEN DEVELOPMENT

Test Driven Development (TDD) although is a proven approach to build automation regression suite around each function and method. This definitely helps in validating the output of each function and make sure function adheres to the developed specifications.

But as per the 2010 report^[1] the success percentage is around 60 only. This might stem up due to incorrect requirement gathering of communication gap between the business owner and the component team.

Hence, there is a need to build an automated suite of functional test cases that can validate the acceptance criteria's set by business analysts or applications owners.

3. ACCEPTANCE TEST

Acceptance tests usually represent the functional end to end tests keeping product or an application in scope

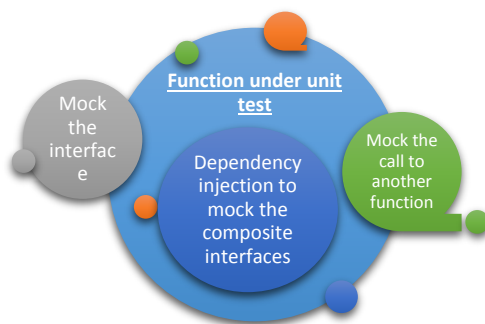
Acceptance tests are driven by the end user requirement of the system or application. These are then reviewed by a designated business analysts, process consultant, and end to end designers. This helps in estimating the usage of the user requirement from strategic view point and defining the granular level requirement of the system or application to be developed. Doing so also

helps to improvise and refine the user requirement. This activity in turn helps in estimating the appropriate size of an application.

4. HOW ACCEPTANCE TESTS ARE DIFFERENT FROM UNIT TESTS

Unit Tests are used to test the latest functionality developed or updated like a method or a function for a given class or an object. The unit tests focus on the lowest possible functionality and focus on the technical aspects. Normally, the scope of unit tests is aligned to the particular function and scenario under development. Database aspects of the incoming/outgoing calls are the key aspects of these unit tests

Acceptance Tests are oriented to capture the acceptance criteria's for the end to end journey of the product(s) or service(s) being offered/rolled out.



In comparison to unit test cases, acceptance tests are dependent on user requirement to ease the way of the working. This also helps to reduce the cycle time of the overall process

It includes the interactions between different modules of the applications, systems, database, user interface, and manual touch points to name a few.

It's advisable to have the appropriate data live like data set up rather than dummy for execution to provide near to realistic results.

5. ACCEPTANCE TEST DRIVEN DEVELOPMENT

Acceptance Tests Driven Development (ATDD) is a collaborative exercise between consultants, designers (to build the acceptance criteria's), testers and developers to first write

acceptance (or functional) test cases for a solution or a product and thereafter use the acceptance test cases for further development.

5.1 Functional / Acceptance Testing

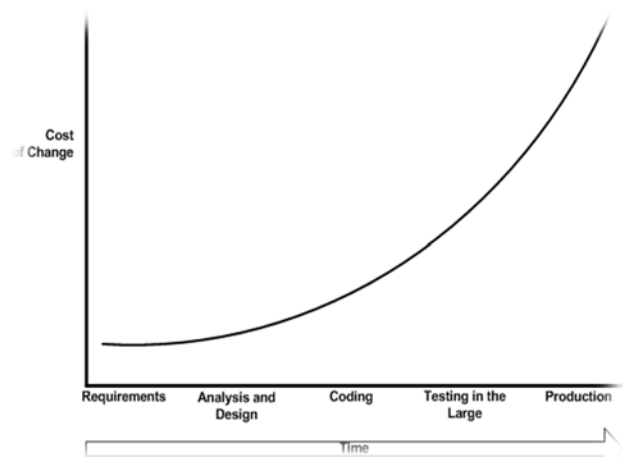
The primary aim of any testing (unit, acceptance or functional) is to identify and resolve defects at the earliest in the SDLC.

IVVT or end to end functional test teams perform functional testing of an application post the development sprint. This approach ensures the quality deliverable of an application being rolled onto a production stack. However this comes with certain limitations:

- Any functional gaps identified outside the development sprint makes it difficult for developers to fix the issue, as there might be cascading impacts on the other deliverables
- Fixing issue at later stages results in costly re-work in terms of
 - Impact analysis
 - Development
 - Re-testing to validate the fix(s)

This may result in delayed deliveries or unmanageable developed code because of hacks being added to fix the defects

- Extra time and effort required to run the regression pack for any major or minor changes in an application



5.2 How ATDD helps

Acceptance test driven development takes an approach to first define and automate the acceptance tests for the piece of work under consideration. Rather than verifying the developed work at end of development cycle, ATDD ensures continuous feedback to the developer on functional gaps resulting in defect free code at end of development cycle.

- Ensures all developers understand the requirements at granular level
- Early feedback to the developers with failed tests during the development cycle
- Since issues are identified during development sprint, the fixes can be arranged quickly in a strategic way
- Complete automated regression suite helps in validating the quality of an application during any further enhancements

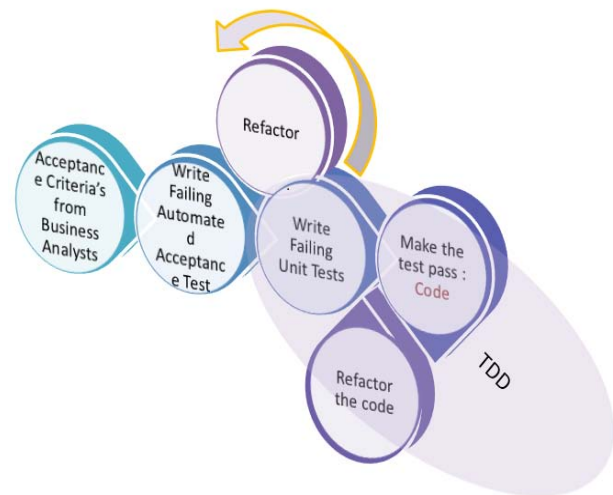
6. ATDD MODEL

With ATDD, developers are supposed to write functional or acceptance tests first, thereafter will proceed with writing of unit test cases that are more atomic with the objective of expecting the defined output from each function or method and finally the actual working code.

Write the Acceptance Test: as a first step, development team write the acceptance test(s) for the given use case(s) before the actual functionality is developed.

This ensures:

- development satisfies the testability criteria
- all the expectations of customers are considered and met as per the standards



Fail the Acceptance Tests:

It's expected that for the above mentioned new acceptance test cases must fail if the functionality is not developed. To ascertain these acceptance test are run on the existing code.

Thus idea behind failing these tests first is to ensure the completeness of end objective once code is developed and test is passed. In case test case(s) is/are passed without developing the code, it essentially means no need of any development.

Write the Unit Tests: After defining the end objective in form of functional (acceptance) test cases, it's important for developers to write the unit tests before the functional code. This helps to ensure that every line of code is testable and has the defined purpose, hence ensuring the best practices. This also helps in eliminating any unwanted lines of code.

While writing unit tests, developers should ensure that unit is smallest entity to be tested for given functionality.

Fail the Unit Tests: Since the functional code is yet not ready, the above unit test must fail. This to ensure that after writing the functional code, the objective of that particular function is met.

Write the functional code: Implement the objective of functional unit and ensure the unit test pass.

Refactor the code (if required) to optimize it further while ensuring no unit test case fails.

The Unit cycle (TDD) is repetitive till the end objective of passing the acceptance test is achieved.

[4]: <http://www.jetbrains.com/teamcity/>
[5]: <http://www.concordion.org/>
[6]: <http://fitnesse.org/>

Finally integrate the build activities and test execution with **continuous integration** server ^{[2][3][4]} to ensure that test cases (both acceptance and unit) got executed with every build.

7. ATDD TOOLS

- ***Concordion***^[5]

Concordion is a java based framework on top of junit runner that helps writing the acceptance criteria's on to HTML page in an assertion mode. This helps in displaying the final results on HTML page with useful documentation highlighting the pass and failures using color code.

The tests can be written in pure business language on HTML page, which is thereafter picked by java classes for analysis and execution of test criteria in HTML's.

Concordion helps evaluating the results on web pages by using web drivers (Selenium).

- ***FitNesse***^[6]

FitNesse is also web page testing framework on top of junit that can receive instructions both from well-defined acceptance criteria's written on top of HTML and wiki text as well.

8. CONCLUSION

With ATDD we found that teams have better understanding of the requirements as it mandates the exact behavior in terms of acceptance criteria's. The improved understanding of requirements results in reduced defect density hence reduced CoQ (Cost of Quality). This also helps in simplifying the need of continual or breakthrough improvement as per changing business requirement. Thus, we consider that ATDD is an effective way of developing an application in the continuous evolving environment.

9. REFERENCES:

[1]: <http://www.drdoobs.com/architecture-and-design/2010-it-project-success-rates/226500046>
[2]: <http://jenkins-ci.org/>
[3]: <http://hudson-ci.org/>