Advanced Science and Technology Letters
Vol.120 (GST 2015), pp.170-174
http://dx.doi.org/10.14257/astl.2015.120.32

Design of Acceptance Test Process through Comparative Analysis of the Test Automation Technique

Jung-Ah Shim[1], Hyun-Jung Kwon[2],Hyun-ju Jung [3] and Moon-Sung Hwang[4*]

[1,2,3,4*]Department of IT Policy and Management, Soongsil University, Seoul 156-743, Korea
[1]rosapoodle@naver.com, [2]musehjkwon@gmail.com, [3]hyunju@hashtag.co.kr, [4*]
ms_hwang@nate.com

Abstract. An acceptance test refers to a test that confirms whether the specified requirements are met. Recent studies on acceptance test automation appear to focus only on easy and simple test automation, while the importance of living documentation for the general application life cycle, which is the ultimate goal, is overlooked. Compared to traditional development methodologies, it points to marked absence of systematic planning and prediction, overhead due to the application of new process and tools, and in turn, a decline in development productivity. This study designed an architecture that can be used repeatedly through the common application of the Agile software. Concrete action plans for automating the acceptance test are presented.

Keywords: Acceptance Test, QA, ATDD, FitNesse

1 Introduction

Though the test is clearly defined, the term acceptance test seems to cause some confusion. Wikipedia defines the term in various contexts, while the definition by XP (Extreme Programming) is closest to what is used in this study. An acceptance test represents the agile software development, especially the function test by the development team at the XP-implementation level, based on the user stories.[1] Unlike the Unit Test, the purpose of acceptance test is to validate business functions, not unit functions. As each business function is likely to be implemented in separate teams and separate environments, the interaction between these business functions is also subjected to an acceptance test. For example, when the business functions are implemented in separate servers, an acceptance test confirms whether those functions interact properly.

ISSN: 2287-1233 ASTL

2 Related Research

The market share of agile methodology has increased as many companies and developers consider or prepare importing this technique. It includes Extreme Programming, Scrum, and test-driven development as a core process of Extreme Programming. [7] The biggest benefits of test-driven development involve a flexible and practical program development process and the availability to develop high quality programs. In general software methodologies, such pressure for a test is why the transition to test-driven development is difficult. Therefore, it can be concluded that unit test tools suitable for test-driven development are required to minimize the pressure of test on developers. This would also allow more flexible response to changes in test cases. Since a unit test is at a low level, once it detects and corrects low-leveled errors, they need to be combined to conduct an integration test in a module group unit. An acceptance test is discriminated from a unit test when it is conducted not only by the programmer or the tester but also by various interested parties including the customers. Table 1 shows the difference between the two tests.

Table 1. Difference between Unit Test and Acceptance Test

	Unit Test	Acceptance Test
Subject Validated	Unit functions	Business functions (requirements)
Conductor	Programmer	Various interested parties such as programmer, tester, and customers

Traditionally, acceptance test was performed directly by the QA (Quality Assurance) team, but the cost increased because too much manpower was inputted to the passive tests. To address this problem, the need for the introduction of test automation tools was raised. Test automation tools are classified by phase in Table 2 below.

Table 2. Test automation tools are classified by phase.

SDLC	Automation Tools	Description
Design Phase	Specification-based test design tool	Test processes, data, drivers, etc. are generated from software specifications.
	Code-based test design tool	Test processes, data, drivers, etc. are generated from source codes.
	Test-based management tool	Supports planning, requirements, and bug tracking management.

Implementation Phase	Static analysis tool	A tool for analyzing without running programs. Analysis of complexity
	Dynamic analysis tool	System status is examined while the program is running.
	Review and inspection tool	Guidelines and rules are inspected by analyzing source codes and design documents.
	Coverage measurement tool	Degree of completion of test case performance
	Performance and load simulation tool	Number of transactions per second is calculated after the occurrence of a system performance load.
Post-Implementation Unit Test Phase	Test performance tool	Performs unit and total system tests before acceptance test

The Agile Software Development process that requires frequent performance of acceptance tests due to short development cycles has a greater need for automating acceptance tests to reduce manpower input and costs.

3 Designing Acceptance Test Automation Architecture

3.1 Concluding Automation Architecture Driver

Non-functional requirements are reviewed through the main process in the concept of "specifics using examples," which is the ideal model of acceptance test-driven development.

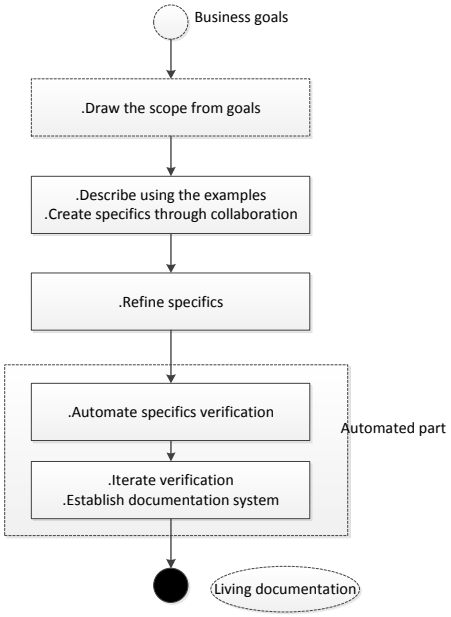

Fig. 1. Main process of the specifics using examples

3.2 Designing and Applying the Process for Constructing the Automation

A process for constructing acceptance test automation was conducted in four stages: establish the strategies, draw requirements, write requirement specifics and test cases, and construct test automation. On stages 2 (draw requirements) and 3 (write requirement specifics and acceptance test cases to validate them), iteration was applied to repeatedly supplement the process through mutual verification. Such repetitive supplement was continued until the specifics clearly form three stages as proposed here: user story, step definition, and example. In addition, iteration was applied in implementing the test code at the test automation construction level through four stages; fixture writing, fixture refactoring, composing test suites, and conducting test. Through such fixture refactoring process, method/class extraction was repeated until it formed the proposed architecture: custom fixture, test utilities, and application driver.

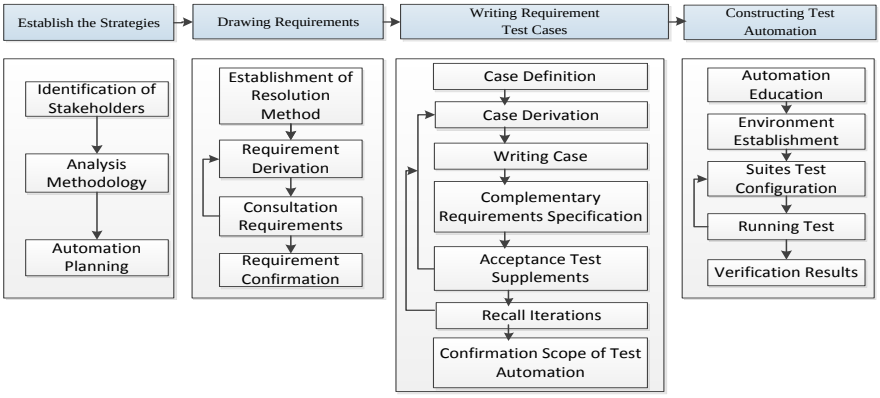

Fig. 2. Requirements / acceptance test automation process for rebuilding

4 Conclusion

Modern software testing methods are designed to define features and functions that business experts desire, growing out of past automation tests to verify development product and find faults at the end of the development cycle [2]. However, such core value is conceptual and lacks in practical application and instances. With many supporting automation tools, this pattern was flooded with application methods in many different dimensions. It was even altered as a simple UI-based acceptance test method. This study proposed an acceptance test automation method for "specifics using example" based on establishing an architecture. The results can serve as a practical basis for typical and repetitive execution of the conceptual Agile acceptance test-driven development methodology to secure fundamental quality.

References

1. https://en.wikipedia.org/wiki/Acceptance_testing
2. Lisa Crispin, Janet Gregory: Agile Testing: A Practical Guide for Testers and Agile team: pp. 47--51 (2012)
3. David Janzen and Hossein Saiedian: Test-Driven Development: Concepts, Taxonomy, and Future Direction: The flagship publication of the IEEE Computer Society, Vol.38, No.9, pp. 1--3 (2005)
4. Hans Wasmus and Hans-Gerhard Gross: Evaluationof Test-Driven Development Delft University of Technology Software Engineering Research Group Technical Report Series.(2007)
5. Charles D. Allison: The simplest unit test tool that could possibly work: Journal of Computing Sciences in Colleges, Vol.23, Issue.1, pp.183-189 (2007)

