Student Experiences with Executable Acceptance Testing

Kris Read, Grigori Melnik, Frank Maurer
Department of Computer Science, University of Calgary
Calgary, Alberta, Canada
{readk, melnik, maurer}@cpsc.ucalgary.ca

Abstract

This report describes experiences of introducing executable acceptance testing in senior software engineering courses. Students in an agile environment completed a five-iteration project with significant portion of requirements specified as executable acceptance tests. Furthermore, students were required to write their test suites for an additional component. Ability to learn and utilize the FIT acceptance testing framework is evaluated to find out if FIT tests can be used to replace requirement documents. The results of a survey of students' perceptions and experiences were encouraging.

1. Introduction

Acceptance testing is a formal technique to ensure that a system satisfies the expectations of the customer who commissioned the software. Acceptance tests verify code against the requirements and act as a type of check of contractual obligation between customer and developer. Acceptance tests can also be used as a measure of progress. These tests are written from the perspective of the user, and test the system as a whole (as opposed to unit testing, which tests technical detail). The motivation for acceptance testing is to demonstrate working functionality rather than to find faults (although faults may be found as a result of acceptance testing). They are traditionally specified using scenarios and performed by quality assurance teams together with the user or representatives thereof (e.g. business analysts). Some drawbacks of the acceptance tests are that they consume a lot of time to manually execute, especially when they need to be executed repeatedly for multiple releases. Manual tests are also prone to human errors. To address these concerns, the technique of executable acceptance testing was introduced. This technique automates the execution of tests by encoding example inputs, expected outputs, and the plan of execution into a test case which can be run automatically. Such executable tests can be executed more frequently, increasing feedback between the customer/user and developer. It is important that an executable test case specification remain understandable by the end user, because it is the end-user who should provide the acceptance criteria and be interested in the results.

There are many tools that can be used to create automated acceptance tests. Some types of tools used include "record-and-playback" testing tools (working on the user interface layer) or scripting languages. Two tools popular among agile practitioners are FIT [10] and FITNesse [11]. FIT is a framework for acceptance testing that encodes the test case (scenario) as a set of human-readable tables. These tables are well suited for representing functional requirements. FIT tables are then executed by "fixtures" which are written by developers to check that their code meets the criteria defined in those tables. These fixtures can be implemented in any programming language. FITNesse is a wiki for defining, organizing and executing FIT Tests. Both FIT and FITNesse are free, open-source tools.

This paper reports on the experiences of software engineering students while learning, reading and writing acceptance tests during the course of their projects. Our objectives during the course of this study were:

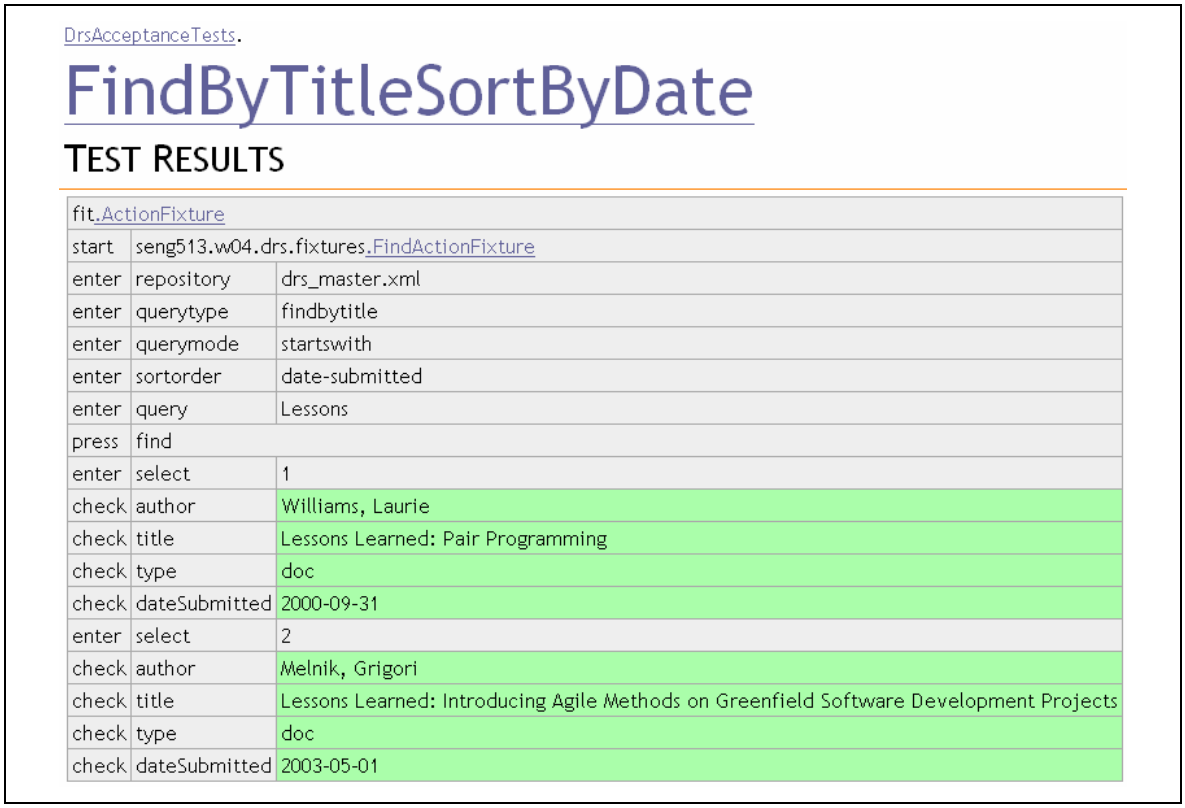
- To educate students about acceptance testing techniques;
- To investigate student perceptions of acceptance testing;
- To determine student opinions about FIT and FITNesse tools.
- To determine if executable acceptance tests can actually be used to communicate requirements.

2. Background

Increasing attention is being drawn to executable acceptance testing within the research community. Specifically, preliminary research is slowly been done by the agile community. Steinberg has looked into how acceptance tests can be used by instructors to clarify programming assignments and by students to check their progress in introductory courses [8]. Andrea discussed an approach involving generating code from acceptance tests specified in a declarative tabular format within Excel spreadsheets [1]. There is an ongoing debate about who should write acceptance tests [7], and the differences between acceptance testing and unit testing has been examined by Rogers [6]. He provides practical advice on defining a common domain language for requirements, helping customer to write acceptance tests, and integrating the acceptance tests into the build process. Watt and Leigh-Fellows described an adaptation of XP style planning that makes acceptance tests central not only to the definition of a story but also central to the process itself. They showed how acceptance testing can be used to drive the entire development process using industrial case studies [9]. Mugridge and Tempero discussed evolution of acceptance tests to improve their clarity for the customer. The approach using tables for acceptance test specification was found to be easier to use than previously developed formats [4]. Moreover, tutorials and peer-to-peer workshops on acceptance testing frameworks and practices have recently been appearing at agile methods related conferences.

The present study is the third in a series of research investigations focused on executable acceptance testing. These studies have spanned a period of two academic years and have involved participants from the University of Calgary and SAIT Polytechnic.

The original study [3] set out to investigate the suitability of FIT for specifying functional requirements from the developer's perspective. This study found that FIT tests describing customer requirements can be easily understood and implemented by a developer with little background. FIT was furthermore found to be a viable format for functional requirements definition based on our criteria (dealing with noise, over-specification, wishful thinking, ambiguity, forward referencing, oversized documents, reader subjectivity, customer uncertainty, and multiple representations).



DrsAcceptanceTests.

FindByTitleSortByDate

TEST RESULTS

fit.ActionFixture		
start	seng513.w04.drs.fixtures.FindActionFixture	
enter	repository	drs_master.xml
enter	querytype	findbytitle
enter	querymode	startswith
enter	sortorder	date-submitted
enter	query	Lessons
press	find	
enter	select	1
check	author	Williams, Laurie
check	title	Lessons Learned: Pair Programming
check	type	doc
check	dateSubmitted	2000-09-31
enter	select	2
check	author	Melnik, Grigori
check	title	Lessons Learned: Introducing Agile Methods on Greenfield Software Development Projects
check	type	doc
check	dateSubmitted	2003-05-01

Figure 1. A Sample FIT test from the DrsAcceptanceTests Suite after execution.

The second study [5] broadened our research objectives to search for patterns in how executable acceptance tests are written. Several patterns emerged, such as incremental addition of passing assertions and a common use of preferred FIT fixture types. We also saw some clear divisions between contexts, such as the relative "fatness" (degree to which fixtures were not refactored) of the fixtures produced being widely disparate. When analyzing patterns in the use of FIT for regression testing, we found that each of the teams in this study decided for themselves when to run suites versus single test cases. Overall, this study supported that subjects were able to interpret and implement FIT test specifications without major problems.

This third study follows up on the previous two by gathering data to support or refute our previous conclusions based on a new perspective: the perceptions of the participants.

3. Context of Study

Acceptance testing was introduced in senior software engineering courses, the primary focus of which was development of Web-based systems. Students were given the opportunity to learn agile practices, including test-driven design, unit testing, and acceptance testing early in the course. This training also included demonstrations of how to use FIT and FitNesse.

The practical component of the course included a series of assignments to design and build a document review system. Each assignment required participants to deliver an increment of working functionality. Assignment One required to design a data model (using XML Schema) and to specify several transformations (using XSLT). In this assignment, students' exposure to FIT was limited to understanding an assignment specification (given in the form of a suite of FIT tests – see Figure 1 for an example) with the assistance of instructor and teacher's assistants. Assignment Two involved building the business logic using Enterprise Java Beans (session beans); Assignment Three – Web-based user interface; Assignment Four – implementation of persistence layer (entity beans). These three assignments did not involve any new acceptance tests (although other types of testing were introduced). In Assignment Five, participants built a Web-service for their document review system. Teams were required to write acceptance tests for the Web-service functionality and then exchanged tests cases with one another. Some teams received acceptance test suites from teams at another institution. In this assignment, we looked into their ability both to write and understand acceptance tests independently.

4. Subjects and Sampling

Students of computer science programs from the University of Calgary (UofC) and the Southern Alberta Institute of Technology (SAIT) participated in the study. All individuals were knowledgeable about programming, however, no individuals had any knowledge of FIT or FitNesse (based on a verbal poll). In Winter 2004 semester, 28 senior undergraduate UofC students from the Web-Based Systems course and 14 students from the Bachelor of Applied Information Systems program at SAIT who were enrolled in the Internet Software Techniques course, took part in the study. Both courses were taught by the same instructor and covered identical topics. In total, 9 teams with 4-6 members were formed. The opt-in survey was distributed to all 42 subjects. The results of the survey were entirely anonymous. It should be noted that this is not a random sample. Therefore, the limitations of the self-selected sample should be considered. Additional qualitative information was informally gathered from in-class polls and during weekly stand-up meetings. Analysis of all raw data was performed subsequent to course evaluation by an impartial party with no knowledge of subject names and had no bearing or effect on the final grades.

5. Analysis

Throughout the semester we were interested in subjects' abilities to interpret and specify executable acceptance tests. In particular, we inquired whether the FIT tests themselves were sufficient specifications of functional requirements. In the survey, students could choose which questions to answer, and therefore response rates (*RR*) are noted where appropriate.

When asked to rate the appropriateness of acceptance tests for defining functional requirements, students on average answered "adequate". Fully one-third answered "good" or "excellent" to this question. No students answered "inappropriate" (Figure 3: Top).

A second question was "Did you require additional written resources to complete the assignment, and if so what resources?" (*RR* =73%). Students responded that no additional written resources were needed in 36% of replies. Some of the resources mentioned included:

more FIT examples, FitNesse tool documentation, with only one person desiring "a better project description". Subjects were surveyed to see if they required additional verbal instruction to complete the assignment (*RR*=70%). Thirty percent of students who replied answered that they did not need any additional verbal help to understand the assignment when specified using FIT. The remainder indicated they had sought TA or instructor assistance at some point.

Asked "would you have preferred to have this assignment specified entirely as prose (text) instead of as FIT acceptance tests?" subjects answered "no" in 16 out of 20 responses. This indicates a clear preference for using user acceptance tests over pure prose requirements specifications. The majority of those who answered "yes" explained that for them FIT was difficult to learn.

Subjects compared defining FIT acceptance tests with writing a specification in prose (*RR*= 83%). On average, students found these tasks to be equal in the level of difficulty. Over one third found FIT to be easier or much easier to use (Figure 3: Middle).

Similarly, subjects compared defining FIT acceptance tests with writing JUnit tests. These were the only two testing frameworks students were exposed to. Again, on average, results indicated that these tasks were equally difficult (Figure 3: Bottom). One third of subjects answered that defining FIT tests was easier or much easier than defining JUnit tests. Twenty-seven out of 30 subjects responded to this question.

Furthermore, we were interested in knowing if subjects believed that someone else could use acceptance tests that they defined to create working software (*RR*=66%). A majority (65%) of responses were affirmative. Only three responses making up 15% indicated that they did not think FIT acceptance tests could be used to communicate the specifications to create working software, and two out of three negative responses indicated an incomplete understanding of FIT or the question asked.

Following up on gathering the level of student's knowledge about FIT, we asked whether better knowledge of FIT acceptance tests would allow subjects to create a better specification. Majority (75%) believed that it was the case (*RR*=80%). Some opinions included "[FIT] is much clearer than writing requirements in prose", and "If one knows FIT acceptance tests first then he/she understands the system well".

Since the subjects randomly exchanged test suites, we inquired if the acceptance test given to them by the other team was sufficient to create the Web service. Obviously, replies to this question varied depending on the quality of specification received. Examples of poor-quality specifications included misuse of FIT which tightly coupled the tests to a specific implementation. In 43% of cases students answered with a definite yes; 26% answered that they managed to create the service but had problems; 31% indicated that the FIT tests they were given were insufficient.

With regard to whether subjects communicated with the other team (who provided the test suite for the required Web service) by means other than acceptance tests (*RR*=73%), nearly half (41%) of respondents indicated that they did not. There are two possible explanations to this: 1) tests were clear enough that no additional communication was required; 2) teams could not or did not attempt to communicate. Of those who did communicate with the other team, common means included email, face-to-face, sharing files, phone, and using an online collaborative space [1].

Lastly, we queried if subjects would choose to use FIT acceptance tests as a means to define functional requirements in the future. More than half (52%) said a definitive "yes" and three more indicated "maybe". This cumulatively means that over two-thirds of respondents would consider using the FIT acceptance testing framework in the future.

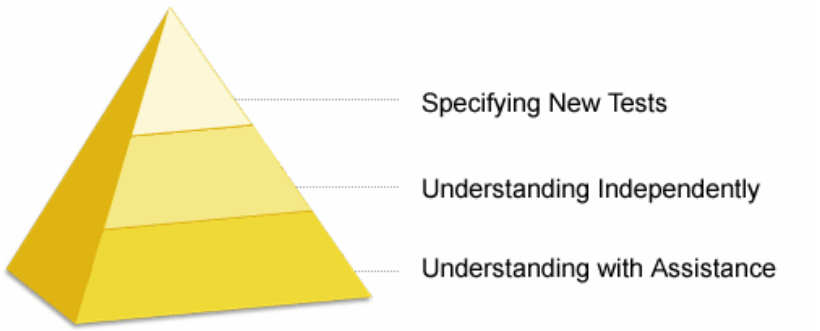
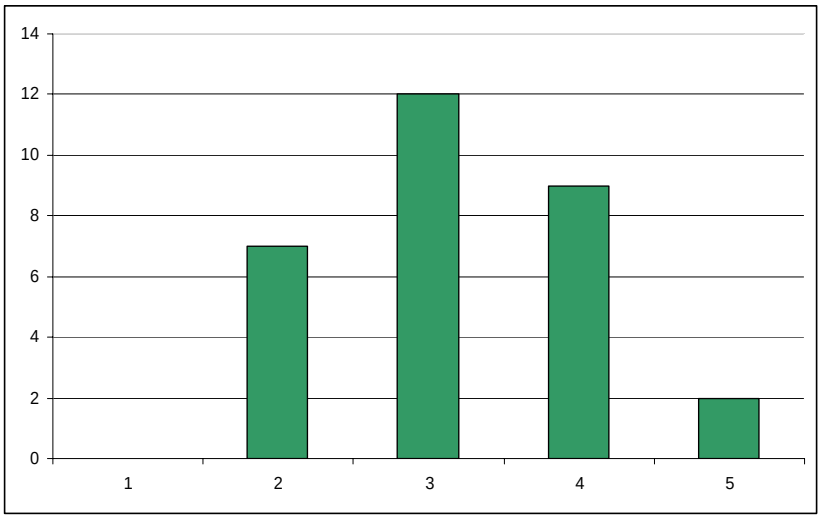
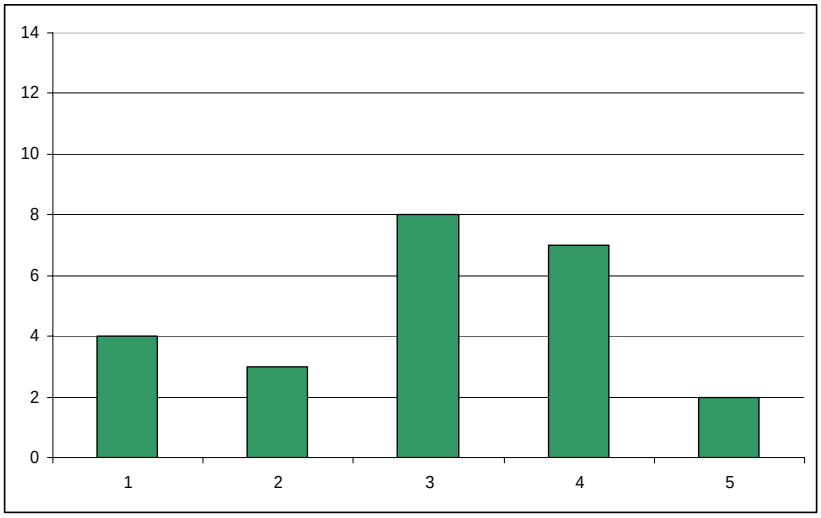


Figure 2. Three levels of Executable Acceptance Testing Comprehension.



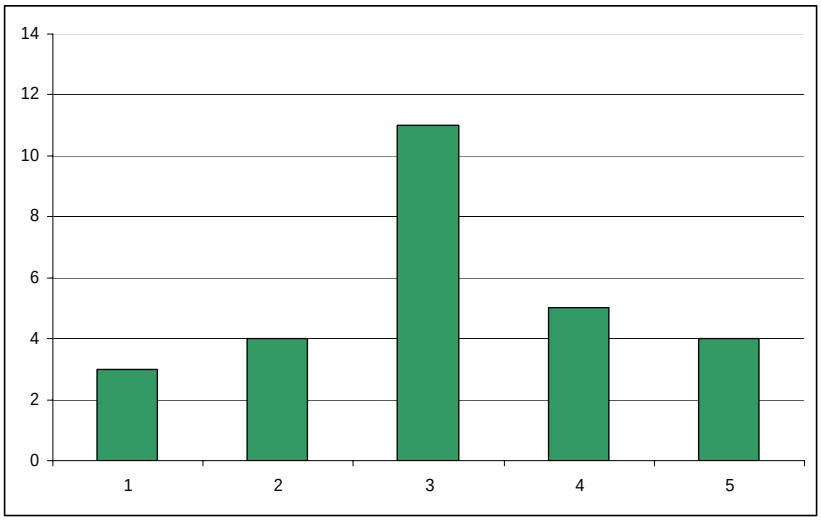
Do you think that FIT acceptance tests are a good method of defining functional requirements?

1 = Poor
5 = Very Helpful



Do you think that defining FIT acceptance tests are easier or more difficult than defining a written specification?

1 = Much Harder
5 = Much Easier



Do you think that defining FIT acceptance tests are easier or more difficult than writing JUnit tests?

1 = Much Harder
5 = Much Easier

Figure 3. Student responses to Likert scale questions. ***1 = Poor/Harder 5 = Excellent/Helpful***

6. Lessons Learned

From the perceptions and experiences gathered, we have identified three levels of understanding exhibited by our subjects (see Figure 2).

The first level of understanding is characterized by being able to read and understand executable acceptance tests with the assistance of a trained expert. At the beginning of the course when introducing FIT through examples and demonstrations, students could quickly grasp the meaning of each test case. This level of understanding will possibly be exhibited by most customers or users who do not have a technical background. FIT in particular is fairly human readable but nonetheless may require assistance to understand depending on the person's background.

The second level of understanding, and one which eclipses the first level, is being able to read and understand acceptance tests independent of outside information sources. This level of understanding requires solid knowledge of acceptance testing, the format used (FIT), and often the ability to interpret, understand and articulate the functional requirements found in the underlying test cases. In our experience, students of information technology are able to reach this level of understanding fairly easily after being trained and given opportunities to practice.

The third and greatest level of understanding is required in order to specify new test cases. Authoring acceptance tests is more difficult than reading and understanding them. Tools like FitNesse make organizing and inputting tests easier. However, tools cannot give one the cognitive ability to judge the quality of an acceptance test. Moreover, writing acceptance tests at a "high-level" with a well defined target and scope requires much practice and experience. A significantly smaller fraction of students were, in our opinion, able to reach this level of understanding (as exhibited in Assignment Five, see Section 3).

7. Conclusion

Overall, results from the perceptions survey were on the whole positive. For most questions responses praised executable acceptance testing or indicated that it was as useful for describing functional requirements as a prose assignment definition. Negative or criticizing comments about FIT and related technologies were in the minority. This study focused on specific technology and a practical assignment, and, therefore, the results may not be applicable to acceptance testing in general. At the very least, this sort of response indicates that more study is needed and that executable acceptance testing is a technology worth pursuing both in the educational arena and as a developer tool.

References

[1] Andrea, J. "Generative Acceptance Testing for Difficult-to-Test Software", *Lecture Notes in Computer Science*, Vol. 3092, Springer-Verlag: 29 – 37, 2004.

[2] Lowell, C. and Stell-Smith, A. "Successful Automation of GUI Driven Acceptance Testing", *Lecture Notes in Computer Science*, Vol. 2675, Springer-Verlag: 331-333, 2003.

[3] Melnik, G., Read, K., Maurer, F. "Suitability of FIT User Acceptance Tests for Specifying Functional Requirements: Developer Perspective", *Proc. XP/Agile Universe 2004, Lecture Notes in Computer Science*, Vol. 3134, Springer-Verlag: 60–72, 2004.

[4] Mugridge, R. and Tempero, E. "Retrofitting an Acceptance Test Framework for Clarity", *Proc Agile Development Conference*, IEEE Press: 92-98, 2003.

[5] Read, K. Melnik, G. Maurer, F. "Examining Usage Patters of the FIT Acceptance Testing Framework", *Proc. XP 2005, Lecture Notes in Computer Science*, Springer-Verlag. 2005.

[6] Rogers, O. "Acceptance Testing vs. Unit Testing: A Developer's Perspective", *Lecture Notes in Computer Science*, Vol. 3134, Springer-Verlag: 22 – 31, 2004.

[7] Sepulveda, C., Marick, B., Mugridge, R., Hussman, D. "Who Should Write Acceptance Tests?", *Lecture Notes in Computer Science*, Vol. 3134, Springer-Verlag: 184 – 185, 2004.

[8] Steinberg, D. "Using Instructor Written Acceptance Tests Using the Fit Framework", *Lecture Notes in Computer Science*, Vol. 2675, Springer-Verlag: 378 – 385, 2003.

[9] Watt, R. and Leigh-Fellows, D. "Acceptance Test Driven Planning", *Lecture Notes in Computer Science*, Vol. 3134, Springer-Verlag: 43 – 49, 2004.

[10] FIT: http://fit.c2.com

[11] FitNesse: http://www.fitnesse.org

[12] MASE: http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=Root.MASE