The Practice of Specifying Requirements Using Executable Acceptance Tests in Computer Science Courses

Grigori Melnik
University of Calgary/SAIT Polytechnic
2500 University Drive NW
Calgary, Alberta, T2N 1N4, Canada
+1-403-210-9710
melnik@cpsc.ucalgary.ca

Frank Maurer
University of Calgary
2500 University Drive NW
Calgary, Alberta, T2N 1N4, Canada
+1-403-220-3531
maurer@cpsc.ucalgary.ca

ABSTRACT

This report describes the practice of using executable acceptance testing for specifying programming assignments in software engineering courses. We summarize experiences from two courses introduced in two academic institutions over four semesters – both from students' and instructors' perspectives. Examples of projects and the discussion of the assignment flows are given. The paper highlights testing as an all-encompassing activity in software development projects. It also contains recommendations for academics thinking of incorporating executable acceptance testing into their courses.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications;
D.2.5 [**Software Engineering**]: Testing

General Terms

Design, Verification

Keywords

Executable acceptance testing, FIT, requirements specification, academic projects

1. INTRODUCTION

Acceptance test is a (formal) test conducted to determine whether or not a system satisfies its acceptance criteria and to enable the user (customer) to determine whether or not to accept the system (as defined in [1] and [9]). Acceptance testing must proceed from the user's perspective (not the developer's). Acceptance tests can be specified in many ways from prose-based user stories to formal languages and scripts. These tests can be executed manually or automatically. Similarly, programming assignments in software engineering courses specify the requirements from the instructor's perspective. Traditionally, they are written in prose. This paper reports on our experiences of using executable acceptance tests for specifying assignments. This approach highlights the role of testing beyond a traditionally limited purpose of detecting failures.

Copyright is held by the author/owner(s).
OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

Testing becomes a more encompassing activity focusing on design and customer interaction. Students start to think of testing early in the project (as opposed to the traditional activity that is done at the end of the project and only if time permits).We share lessons learnt and encourage educators to consider using this technique in their courses, as early as even in the first semester courses. Specifically, we discuss the use of the FIT acceptance testing framework for communicating and testing project requirements in two software engineering courses.

2. FIT FRAMEWORK AND FITNESSE

FIT is an open-source, multi-lingual framework for acceptance testing [5]. It allows specifying acceptance tests in the form of tables. FIT tables can be written in various common formats – Word, Excel, HTML, Wiki[1]. There are several test table styles (Table 1). To be interpreted (executed), these tables require "fixtures", which are built by developers (in any language that FIT execution engine supports). The fixtures are normally written as dispatchers of calls to the business logic of the real system. The end result is an "executable specification" [12].

Fitnesse [6] combines the ideas of FIT (easy edit and execution of acceptance tests) and Wiki (open collaborative space) and allows teams to collaboratively specify test tables and run them through a

Table 1. Common FIT fixtures

Fixture Type	Description
RowFixture	Examines an order-independent set of values from a query.
ColumnFixture	Represents inputs and outputs in a series of rows and columns.
ActionFixture	Emulates a series of actions or events in a state-specific machine and checks to ensure the desired state is reached.
CommandLineFixture	Executes shell commands in multiple threads
TableFixture	Base fixture type allowing users to create custom table formats.

[1] Essentially, any document format that supports tables and can be converted into HTML.

Specification

1. Design a data model (as a DTD or an XML Schema, or, likely, a set of DTDs/XML Schemas) for the artifacts to be used by the DocumentReviewSystem. Concentrate on "Document submission/update" and "Document review" tasks for now.
2. Build XSLT sheet(s) that when applied to an instance of so's repository will produce a subset of so's. As a minimum, queries and three query modes specified in DrsAssignmentOneAcceptanceTests must be supported by your model and XSLT sheets.
3. Create additional FIT tests to completely cover functionality of the queries.

Setup files

drs_master.xml - a sample repository against which the FIT tests were written

DrsAssignmentOneAcceptanceTests.zip - FIT tests, unzip them into FITNESSE_HOME\FitNesseRoot\ directory.

Figure 1. Assignment one specification in Web-based systems course.

Wiki page. Anyone can contribute content to the site without knowledge of HTML or programming technologies.

Ideally, any automated tests should fit into the existing build process. FIT is a command-line tool and Fitnesse can also be run in the command line mode. It allows them both to be included in the build scripts (such as Ant).

3. COURSE AND STUDENT PROFILES

We used executable acceptance testing for specifying assignments in junior and senior courses in two academic institutions over four semesters (Fall 2003 – Winter 2005). Course descriptions and student profiles are provided below.

3.1 Software Testing and Maintenance

Software Testing and Maintenance course is offered in the first semester[2] of Bachelor of Applied Information Systems[3] program at SAIT Polytechnic. This is a required course for students in Software Engineering and Information Systems Development majors. A variety of techniques and tools are introduced including unit testing, integration testing, GUI testing, user acceptance testing, performance testing, mock objects, automated build tools, and continuous integration. Students are responsible for both specifying the test cases and implementing those using testing frameworks with the primary goal of building a quality product.

Students in this program generally belong to one of two groups: 1) full-time learners who entered the program immediately after finishing their college two-year diploma program with a solid knowledge of programming languages, design, and development techniques; 2) part-time adult learners normally employed in the field and taking the program to upgrade their knowledge and to obtain a degree. They work on the project assignments in teams (normally teams of 4) and this mix of less-experienced with more-experienced students creates a vibrant team environment.

This course was originally offered in the second semester of the program, but was moved to the first one. This change has positively affected the level of preparation for the project courses in the following semesters because students are already familiar with common testing techniques, automatic build tools, version control, and collaboration systems. As a result, students are expected to provide test drivers for all future code they write. The doctrine of merciless testing had been "engraved" in students' minds.

3.2 Senior Web-Based Systems Course

Web-Based Systems is a senior course taught by one of the authors at the University of Calgary (majority in Computer Science major and about 25% in Electrical Engineering, Environment Design, Economics, and other majors) and at SAIT Polytechnic (Software Engineering and Information Systems Development majors). The course gives an overview on a broad range of methods and techniques for building Web-based systems, including XML, XSL, J2EE, and Web services[4].

The course includes comprehensive hands-on software development assignments done in teams of 4-6 students. Assignments utilizing the J2EE framework are designed to deepen the understanding of the introduced technologies and design patterns. Students are encouraged to follow agile principles. Code reuse is strongly emphasized. The final exam consists of developing a small Web-based system and is done online – the students have access to all their learning resources and projects and must deliver clean code that works.

4. COURSE CONTEXTS

4.1 Web-Based Systems Development

In the Winter 2004, the senior Web-Based Systems course project required students to build an online document review system. For the first assignment[5], students were required to work on only a partial implementation concentrating on the submission and review tasks. The only information provided in terms of project requirements was:

1. An outline of the system no more detailed than that given above.
2. A subset of functional requirements to be implemented (Figure 1).
3. A suite of FIT tests.

Requirements in the FIT test suite can be described generally as sorting and filtering tasks for a sample XML repository. Our

[2] In essence, it is the fifth semester, as the students are required to have graduated from a computer technology diploma program (additional two years of prior study).

[3] Similar to a BTech degree.

[4] More information, including course description, outline and assignments available at http://mase.cpsc.ucalgary.ca/seng513

[5] http://mase.cpsc.ucalgary.ca/EB/Wiki.jsp?page=Root.SENG513w04AssignmentOne

provided suite initially consisted of 39 test cases and 657 assertions. In addition to developing the code necessary to pass these acceptance tests, participants were required to extend the existing suite to cover any additional sorting or filtering features associated with their model. An example FIT Test finding a document by author, with results sorted by date submitted is shown in Figure 2.

Participants were given two weeks (unsupervised) to implement these features using XML, XSLT, Java and the Java API for XML Processing (JAXP). A common online experience base was set up and all students could utilize and contribute to this knowledge repository. An iteration planning tool and source code management system were available to all teams if desired.

The learning objectives for the first assignment were to master XML Schema for document modeling and practice XML processing (with XSLT and JAXP). Teams had between 1 and 1.5 weeks to master FIT in addition to implementing the necessary functionality (depending on if they were from SAIT Polytechnic or the University of Calgary). All student teams were able to interpret and understand FIT tests. Median passing rate of instructor-provided test cases was 90% in the submitted assignments. It is important to note that FIT was introduced to students for the first time. This suggests that the FIT learning curve is not very steep.

FIT acceptance tests were used again later in the course in another assignment, in which teams had to Web-service-enable selected system functionality. In the second part of this assignment, teams from UofC and SAIT Polytechnic randomly exchanged their acceptance suites and had to code against those.

4.2 Software Testing and Maintenance

The Software Testing and Maintenance course at SAIT Polytechnic is designed to introduce testing practices around a development project. We emphasized the synergy of testing and development activities. For this course, we typically pick some known game for a project. In Fall 2004, we used the game of Rocket Mania [10]. In this game, the player is required to rotate fuses on the game board to connect them into complete paths that will launch rockets. The strategy is to launch as many rockets at a time as possible resulting into additional bonuses In order to go on to the next level, a certain number of rockets (which increases with each level) must be launched in a given period of time. The game ends when the player fails to launch a required number of new rockets.

Course assignments follow a progressive approach (see Table 2). The teams were given an initial suite of acceptance tests specified by the instructor. They were responsible for implementing all necessary fixtures to get all tests passed. Teams were also required to specify test cases for the 6 tests listed in the suite but not specified by the instructor and getting those to pass as well. In addition, students were encouraged to specify any additional tests they would deem to be useful for two remaining categories: Traversing with the Rockets Upgraded and Adding Bonus Elements. Figure 3 shows the suite of acceptance tests and Figure 5 shows a sample test case for traversing the board with a single path formed, removed and new fuses appeared. Notice in this case, several fixtures are used and students implemented interactions between fixtures. Teams had two and a half weeks to master FIT and to complete the assignment.

5. LESSONS LEARNT

Our experiences with utilizing executable acceptance testing in courses of various levels and in different application domains, clearly manifest an opportunity for executable acceptance testing to be used in a Computer Science course of any context. It does not affect the pace of the course or the amount of the material covered. A two-hour lecture and a demonstration are sufficient to get students ready. Alternatively, it can be introduced by teaching assistants in a single lab or tutorial session.

Moreover, FIT, in our opinion, can be incorporated in any class regardless of the development methodology. In both courses, students commented that concentrating more on tests (including

fit.ActionFixture		
start	seng513.w04.drs.fixtures.FindActionFixture	
enter	repository	drs_master.xml
enter	querytype	findbyauthor
enter	querymode	startswith
enter	sortorder	date
enter	query	Melnik
press	find	
enter	select	1
check	author	Melnik, Grigori
check	author	Read, Kristopher
check	title	Customer Collaboration in DSDM Projects
check	type	doc
check	dateSubmitted	2002-05-31
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 2. Sample FIT test from the DrsAcceptanceTests Suite after execution.

acceptance tests) contributed positively to their learning: learning of the topic and learning about assigned projects.

When incorporating acceptance testing in their assignments and projects, the following are three possible strategies for instructors, that we have tried in our courses with variable degree of success.

Strategy 1: Instructor provides a complete acceptance test suite and uses it as the main assignment specification. Students are required to implement logic and test fixtures to make the suite pass. From the perspective of software engineering, a possible downside of this is that though a formal specification (in the form of acceptance tests) is very precise and makes it easier for students to implement the assignment, it may actually be of disservice to them. One may suggest that students are put into the clean, precise, comforting world of the test suite, prepared by the instructor, instead of facing the tricky software engineering problems (such as analysis and articulation of requirements in the precise form) and building their own test suite. This would not be a problem unless the analysis is one of the learning objectives of the course. If the goal is to train students in some technology, framework or development patterns, then Strategy 1 is appropriate. Strategies 2 and 3 address this possible shortcoming.

Strategy 2: Instructor specifies an incomplete acceptance test suite and requires students to extend it. This way they learn by example and also have to analyze the problem in order to write their own test cases. Also, this strategy is closer to the real world as it highlights a notorious problem in the industry when the test suites are often incomplete.

Strategy 3: Instructor assigns the task of writing acceptance tests to teams themselves and then exchanges the suites between teams. Students are responsible for analyzing the problem and specifying the test cases, for interpreting the test cases obtained from another team and implementing required functionality to pass the tests. This is trickier to implement due to the logistics of the exercise but not impossible. In our experience, this approach was less successful. Possible explanations for this are: 1) at the end of the semester, less time could be devoted to the assignments and 2) little or no face-to-face communication occurred between the customer (in this case the team that wrote the FIT tests) and the development team (that had to implement them). In a survey distributed at the end of the Web-Based Systems course, several students commented on the quality of the acceptance test suite they have received: "[The other team] made it (FIT) too close to their model. Our tests were more generic." and "The other acceptance tests were a mess. Misspellings, errors." Nevertheless, twelve out of 21 students who responded to the question of whether the acceptance tests given to you by the other team were sufficient to create the Web service, replied positively, with two students even indicating that "it was very easy".

Additional observations and lessons learnt include:

- Several students in the junior course on software testing and maintenance initially objected to the practice of doing both – writing test cases and implementing code. They believed a testing course should be purely about testing tools and testing techniques. They remained skeptical during the first couple of iterations (assignments), but soon recognized the value of doing both testing and development. Students have realized that testing is much more than just verification and validation. Based on the informal feedback at the end of the course, they have started to appreciate testing as a way of specifying course requirements and guiding their systems design.
- Overall, based on a survey, majority of students suggested that FIT adequately describes the requirements (78%). When asked "would you have preferred to have this assignment specified entirely as prose (text) instead of as FIT acceptance tests?", 80% of students answered "no". This indicates a clear preference for using executable acceptance tests over pure prose for requirements specifications.

Table 2. Course assignment descriptions and new techniques allocations

Assignment	Brief description	New techniques introduced and practiced	Frameworks/tools used
①	- Randomizer to arrange game fuses on the board. - FilteredRandomizer that reduces the likelihood of certain fuse by a certain percentage. This is used in the logic engine of the game, when we the difficulty level increases and certain pieces should appear less frequently than others.	- Equivalence partitioning, - Boundary-value Analysis, - Unit testing	- JUnit
②	- BoardTraverser engine, - Single board traversals, - No rotations, - No fuse replacements, - No bonuses	- Test-first design (TDD), - More unit testing, - Refactoring	- JUnit
③	- Using BoardTraverser engine from the A2, build the logic component for removing (burning) the complete paths and replacing them with new pieces, - Score calculator, - Rotations, - No bonuses	- Executable acceptance testing	- FIT/ Fitnesse
④	- GUI version of the game, - No animation	- Data-driven GUI testing	- Jemmy
⑤	- Networked version of the game (for 2 players), - Players take turns.	- Mock objects - Automated build scripts	- EasyMock - Ant

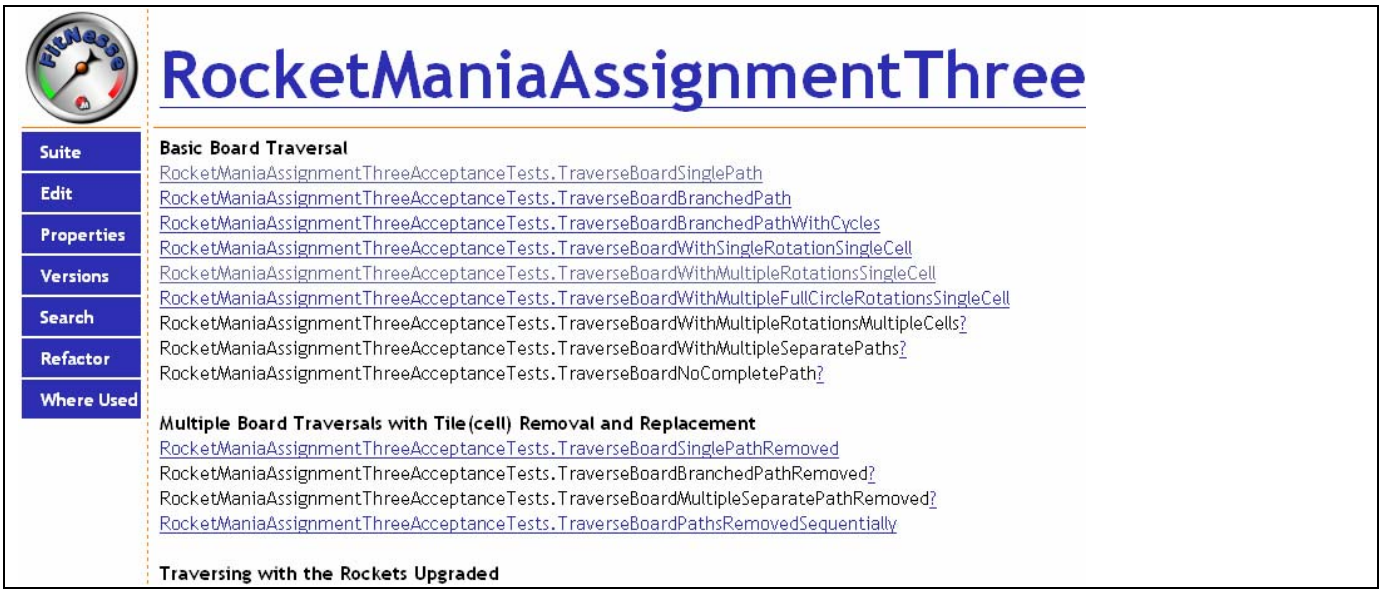

Figure 3. Fragment of the FIT Test Suite for Rocket Mania game in Software Testing and Maintenance course.

- The simplicity of the FIT framework and Fitnesse engine allows for a quick and easy setup. We have used it equally well on local workstations or a centralized server. Test case porting is as easy as copying a subdirectory with all test cases into the FitNesseRoot directory and creating a link to it from one of the existing pages. Submissions can be organized via cvs or simply by asking students to zip all subdirectories of the test suite and add the file to the source code submission. Automating build process is helpful and the execution of FIT acceptance tests can be coded as one of the tasks.
- The support for the framework and the tool is growing. Recently, Mugridge has released the FitLibrary [8] with several new fixtures and useful extensions. It includes DoFixture, an alternative, compact fixture to writing workflow tests which are more understandable. There are also implementations of FIT for C#, C++. Delphi, Perl, Ruby and Python. Undoubtedly, more languages will be supported as popularity of the framework grows.
- In our courses, one of the rules of engagement is that only the code that passes full regression testing can be checked into the repository. Most students began to appreciate this practice as they were able to confidently rely on clean code in the repository.
- Iterative development and "clean code that works" required students to fix all bugs and code "smells" identified by the instructor or TAs in the previous iteration.
- Testing was made an all-encompassing activity that simply could not be ignored. Test drivers that accompany the project code account for at least 30% of the grade. Even though some students failed to see the value of tests at the beginning, they have later admitted that they were "glad they had them as the project progressed". We introduced and encouraged acceptance-test-driven development (test-first design). However, there is no way to control and to enforce it. Not everyone embraced the test-driven approach. Regardless whether students practice test-first or test-last, having a grade component for tests and all-tests-pass-before-check-in policy strengthens the testing culture in the teams. Though one may suggest that this culture will not last without additional reinforcement, the evidence from the graduates currently working in the field indicates preference of many to follow the practice of testing early and often.
- In the process of designing acceptance tests for the board traversal algorithm of the Rocket mania game, we have seen an interesting byproduct that we have not foreseen. There was a discussion among students and the instructor about the way the game board and actions should be represented in the FIT table. Originally, it was proposed to represent each fuse by 4 bits, representing available connections on four sides of the fuse (North, East, South, West). Doing so would have complicated the way of specifying the tests by the customer. We adopted the use of box-drawing characters (Unicode 2500- 253C). This way, the process of writing the tests for complete fuses became visual and easy to follow (see Figure 4). Essentially, a new, domain-specific notation was produced as a result of this exercise.
- In the Web-based Systems project, students were utilizing several fixtures (in the example depicted by Figure 4, TableFixture and ActionFixture were used) and they had to research and implement mechanisms for passing data between the fixtures. Most teams accomplished the task skillfully.
- Many students considered the FIT suite summary as their progress dashboard. Since Fitnesse and the tests were hosted of a centralized server, anyone (including the instructor) could see the progress made at any time.
- This may have also produced a deceptive sense of security – "if the suite passes, my code is good". It is important to remind students to think outside of the box and explore beyond the provided test suite.

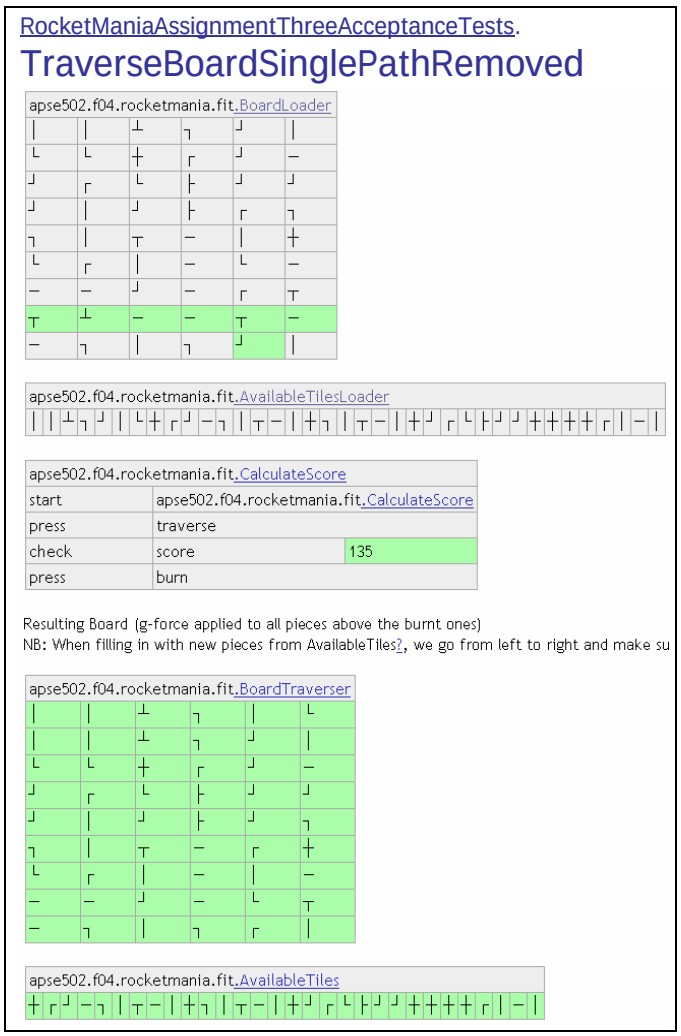

Figure 4. Sample FIT test case with test results from the Rocket Mania acceptance suite.

- We have observed the FIT fixture code produced by students was generally "fat" and contained all required functionality. UofC teams produced fatter fixtures than SAIT teams. This can be explained by the fact that there was little external motivation for the UofC students to refactor their code. Even in the last assignment, when students knew about the test suite exchange, most teams ended up with fat fixtures. This is understandable since no explicit requirement to refactor fixtures was given and there was no strong reason for students to move their logic to another location (outside of FIT code). Conversely, at SAIT students had already implemented business logic in two previous iterations, and were applying FIT to existing code as it was under development, thus producing a more reusable, loosely-coupled code.
- Since executable acceptance tests as requirements are less prone to the sins of traditional requirements: ambiguity, noise, multiple representations, and uncertainty, we have experienced fewer student inquiries of the clarification nature.
- Test-Driven Development/Test-first design practice has drawn significant attention in recent years with the development of agile methods [2], [3]. Engaging in it leads to designs that are well factored and more amenable to testing. Introducing Acceptance-Test-Driven-Development in computer science courses makes students think about their designs from a testability perspective. When writing executable acceptance tests, students also learn how express requirements in a precise, unequivocal manner. This is an important skill for software developers.
- Most importantly, utilizing executable acceptance testing for course assignment specification forces student to think about testing and to practice testing early.

6. SUMMARY

We have described the practice of specifying requirements using executable acceptance testing in academic settings. Based on our experiences of using it for the last four semesters, we find the practice straightforward to implement regardless of the course context. It introduces testing early and makes it an all-encompassing activity. A merciless testing mantra (inspired by both acceptance and unit testing) cultivates the discipline and accountability among software engineering students. This, hopefully, will bring fruitful results as the future graduates may reduce the horrendous failure rate of software projects in the industry [4]. We encourage more software engineering educators to try it out.

7. REFERENCES

[1] Acceptance Test. Online, last retrieved Mar 17, 2005: http://c2.com/cgi/wiki?AcceptanceTest

[2] Astels, D. Test-Driven Development: A Practical Guide. Prentice Hall, 2003.

[3] Beck, K. Test Driven Development: By Example. Addison-Wesley, 2003.

[4] Chaos Report. The Standish Group, West Yarmouth, MA, 1995, 1997, 1999, 2001, 2003.

[5] Cunningham, W. Fit: Framework for Integrated Test. Online, last retrieved Mar 17, 2005: http://fit.c2.com/

[6] Fitnesse. Online, last retrieved Mar 17, 2005: http://fitnesse.org

[7] Maximillien, M. and Williams, L., "Assessing Test-Driven Development at IBM", International Conference on Software Engineering, May 2003.

[8] Mugridge, R. The Fit Library. Online, last retrieved June 25, 2005: http://fitlibrary.sourceforge.net

[9] Perry, W. Effective Methods for Software Testing, 2/e, John Wiley & Sons: New York, NY, 2000.

[10] RocketMania. Online, last retrieved Jan 25, 2005: http://games.yahoo.com/games/downloads/rm.html

[11] Shepart, T. et al. "More Testing Should Be Taught". Communications of the ACM, Vol. 44, pp.103-108, 2001.

[12] Shore, J. Introduction to Fit. Online, last retrieved Jul 14, 2005: http://fit.c2.com/wiki.cgi?IntroductionToFit