Teaching Acceptance Testing in Contexts of Web Systems Development and Game Programming

Grigori Melnik

University of Calgary/ SAIT
Calgary, Alberta, Canada
403-210-9710
melnik@cpsc.ucalgary.ca

Abstract. The paper discusses the role of user acceptance testing and describes the experiences of introducing it into different courses from both students' and instructors' perspectives. Examples of assigned projects are given. The paper highlights testing as an all-encompassing activity in an academic setting. It also contains recommendations for academics thinking of incorporating user acceptance testing into their courses.

1 Introduction

Nowadays, software testing is no longer considered to be a stage of the development life cycle that is done after software construction is completed, with a limited purpose of detecting failures. However, in academic curricula, testing is not always given adequate attention. Even if the role of testing may be discussed in the introductory software engineering courses, the software testing practices themselves are only covered in senior and graduate courses. As a result, students tend to think about testing as something that is done at the end and only if time permits.

This paper reports on our experiences of making testing an all-encompassing activity. We encourage educators to introduce testing in software engineering curricula as early as in the first semester of study, consequently making testing more of a requirement specification technique than a traditional pure verification and validation activity. Specifically, we discuss the use of the FIT acceptance testing framework for communicating and testing project requirements in two software engineering courses.

2 Executable Acceptance Testing

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 [2]). 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. Obviously, automating acceptance test is highly desirable since manual acceptance testing is slow

and expensive. At the same time, making the requirements too formal pushes the customer away from the process. Writing an automated script is a typical programming activity. Languages commonly used for this – Perl, Python, Ruby and Tcl – remain in the toolkit of people competent in coding. Commercial tools (e.g. Rational Robot, Mercury WinRunner) and scripting environments (e.g.SQABasic and VU) exist and are supposed to simplify the task of script/macro creation. However, they are not easy to set up and the licenses are expensive. These tools are typically used by testing professionals and not customers. These tools generally require a system to be in place to test against and, inevitably, place user acceptance testing fairly late in the development life cycle.

3 FIT and Fitnesse

It is possible to begin acceptance testing early, to do it frequently and inexpensively and to make it accessible so that the customer understands easily. One such framework is FIT [5]. It allows customers to specify acceptance tests in the form of tables. FIT tables can be written in various formats ordinarily familiar to the customer – Word, Excel, HTML, Wiki[1]. There are several test table styles (Table 1). To be interpreted (executed), these tables require test fixture implementation to be built by developers (in any language that FIT's execution engine supports[2]). The fixtures are normally written as dispatchers of calls to the business logic of the real system. The end result is an "executable specification" [6].

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.
RowEntryFixture	Special case of ColumnFixture that provides a hook to add data to a dataset.
CommandLineFixture	Executes shell commands in multiple threads.
TableFixture	Base fixture type allowing users to create custom table formats.

Fitnesse [4] 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 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 be run in the command line mode. It allows them to be included in the build scripts (such as Ant).

---

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

[2] Several FIT engines we are familiar with support Java, C#, Perl and Python. Undoubtedly, more languages will be supported as popularity of the framework grows.

4 Course and Student Profiles

We introduced user acceptance testing, FIT and Fitnesse in junior and senior courses in two academic institutions over three semesters (Fall 2003 – Fall 2004). Course descriptions and student profiles are provided below.

4.1 Junior Software Testing and Maintenance Course

Software Testing and Maintenance course is offered in the first semester of Bachelor of Applied Information Systems[3] program at SAIT. 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. The distinguishing characteristic of this course is its practical focus. 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 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 an Applied Bachelor's 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 in fall 2002. 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.

4.2 Senior Web-Based Systems Course

Web-Based Systems is a senior course taught by the author 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 (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:

- Towards Semantic Web: HTML, XHTML, XML, RDF.
- XML Document Modeling (DTD, XML Schema)
- Processing XML with Java (DOM, SAX, JAXP)
- XML Transformations (XSLT)

[3] Similar to BTech degree.

- Client-side technologies (CSS, XSL, JavaScript, DHTML, XForms)
- Building business logic (Session Enterprise Java Beans)
- Controllers for business logic (servlets, filters, plus testing with Cactus)
- Views for business logic (JSPs, taglibs)
- Persisting Web data to the back-end (Call-level interfaces, JDBC, JDO, Entity EJBs, binary and XML serialization)
- Session/user management and personalization
- Messaging middleware
- Interoperability across heterogeneous environments
- Web services (SOAP, WSDL, UDDI)
- Web-enabling legacy applications
- Web usability issues
- Testing of Web-based systems (JUnit, FIT, HttpUnit, JMeter, Cactus, Mock Objects)

It includes comprehensive hands-on software development assignments done in teams of 4-6 students. Assignments utilizing J2EE framework are designed to deepen the understanding of the technologies. 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.

5 The Culture of Testing

In all courses mentioned above, testing is an all-encompassing activity that cannot be ignored. Even in the non-testing specific courses, test drivers that accompany the project code account for at least 30% of the grade. Even though some students fail to see the value of tests at the beginning, they are glad they have them as the project progresses.

In the past, once per semester, we swapped the code bases and made the teams work in the maintenance mode to enhance the code given to them. This is when the true appreciation for testing could be seen. Students' informal comments indicated that "sacrificing tests at the beginning resulted in much more grief later on".

We introduce and encourage test-driven development (test-first design) in all courses. However, there is no way to control and to enforce it. Not everyone embraces the test-driven approach of writing tests first, seeing them fail and then making them pass. Often, the habit of writing code first prevails and students either write tests at the end or even leave certain code without test coverage.

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 enjoy this practice as they were able to confidently rely on clean code in the repository.

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.

6 Acceptance Testing in the Context of Web-based System Development

In the winter 2004, Senior Web-Based Systems course project required students to build an online document review systems (DRS). This system allows users to submit, edit, review and manage documents (articles, reports, code, graphics artifacts etc.) called submission objects (so). These features are selectively available to three types of users: Authors, Reviewers, and Administrators. More specifically, administrators can create repositories with properties such as: title of the repository, location of the repository, allowed file formats, time intervals, submission categories, review criteria, and designated reviewers for each item. Administrators can also create new repositories based on existing ones. Authors have the ability to submit and update multiple documents with data including title, authors, affiliations, category, keywords, abstract, contact information and bios, file format, and access permissions. Reviewers can list submissions assigned to them, and refine these results based on document properties. Individual documents can be reviewed and ranked, with recommendations (accept, accept with changes, reject, etc) and comments. Forms can be submitted incomplete (as drafts) and finished at a later time.

For the first assignment[4], 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 (Fig. 1).
3. A suite of FIT tests (Fig. 2)

Requirements in the FIT test suite can be described generally as sorting and filtering tasks for a sample XML repository. Our 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 title, with results sorted by date submitted is shown in Fig. 3.

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.

All student teams were able to interpret and understand FIT tests. On average, 90% of instructor-provided tests passed in the submitted assignments. It is important to note that in the winter 2004 semester, FIT was introduced to students for the first time.

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

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

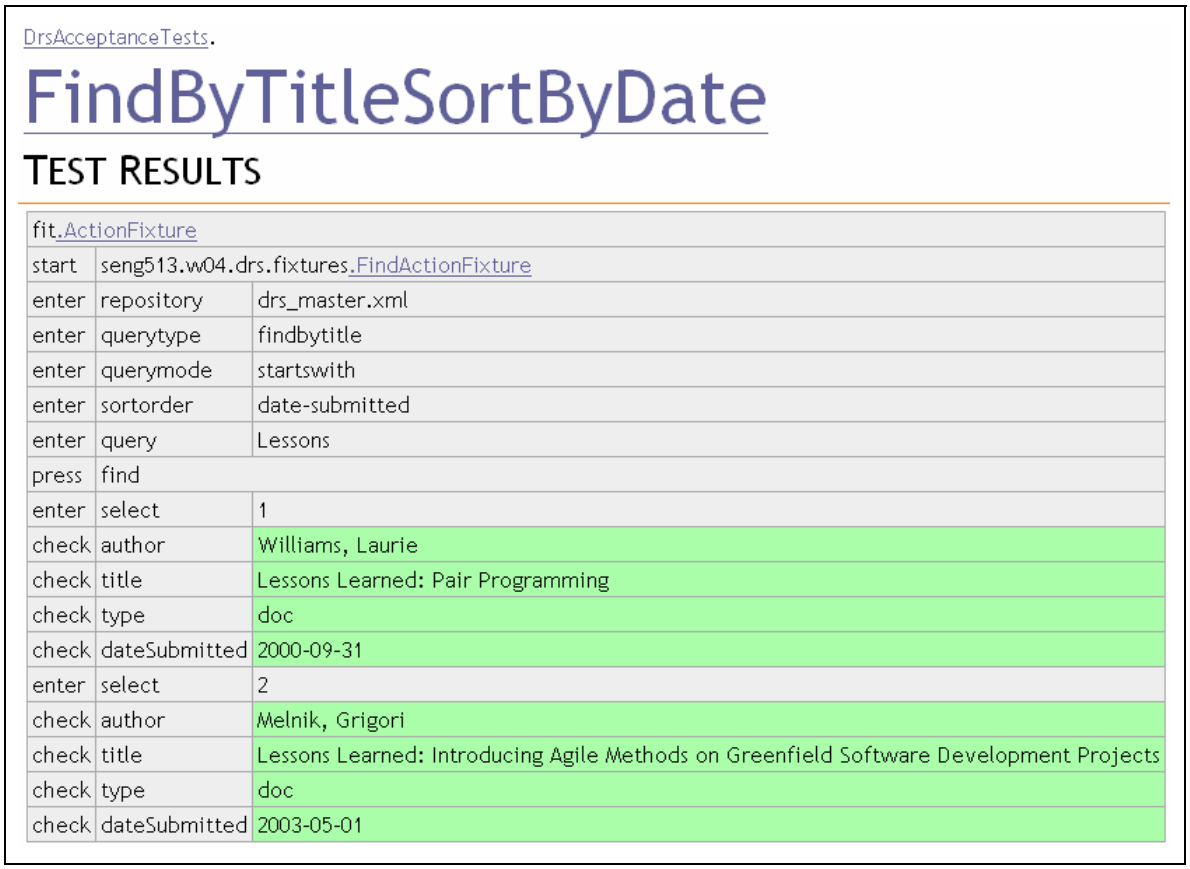
drs_master.xml - a sample repository against which the FIT tests were written

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

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

Figure 2. Partial FIT Test Suite for Document Review System. The suite contains test cases and can be executed. For example, the test FindByAuthorSortByDate results in a unsorted list of items matching an author name.

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 or the University of Calgary). This suggests that the FIT learning curve is not prohibitively steep.



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 3. A Sample FIT test from the DrsAcceptanceTests Suite after execution.

FIT acceptance tests were used again in Assignment 5, in which teams had to Web-service enable selected DRS functionality, specifically:

- list of authors (for a specified document repository)
- find all so's for a given author name (across all repositories)
- give the name of the most active author (the one who has the most so's across all repositories)
- list all distinct countries of author origin (across all repositories).

Teams ended up specifying between 5 and 94 test cases.

In the second part of the assignment, teams from UofC and SAIT randomly exchanged their acceptance suites and had to code against those. This did not go as well as in assignment one. Possible explanation for this may be that at the end of the semester, less time could be devoted to

the assignments and that 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) occurred.

In a survey distributed at the end of the course, a 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 user acceptance tests over pure prose for requirements specifications. Overall, students' perceptions of Fitnesse were also positive.

7 Acceptance Testing in the Context of Game Programming

The Software Testing and Maintenance course at SAIT is structured to introduce testing practices around a development project. We emphasized the synergy of testing and development activities. Several students initially objected to this – 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 acknowledged the value of doing both testing and development.

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, - BVA, - 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	- Automated user 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 4. FIT Test Suite for Rocket Mania game in Software Testing and Maintenance course.

Students have realized that testing is much more than just verification and validation. They've started to appreciate testing as a way of specifying course requirements and guiding their systems design. Also, student teams were enthusiastic about the project because they were delivering pieces of potentially-shippable code. The level of motivation was incredibly high.

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.

For this course, we typically pick some common game for a project. In fall 2004, we used the game of Rocket Mania[5] [7]. 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 (coins, diamonds etc.) Coins are needed to upgrade the rockets. Diamonds give additional points. 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 acquire new rockets.

Course assignments follow a progressive approach (see Table 2).

The teams were given an initial suite of user acceptance tests specified by the client (instructor in this case). 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

[5] PipeDream is a variant of the same game.

RocketManiaAssignmentThreeAcceptanceTests.

TraverseBoardWithMultipleRotationsSingleCell

apse502.f04.rocketmania.fit.BoardTraverser					
│	│	┴	┐	┘	│
└	└	┼	┌	┘	─
┘	┌	└	├	┘	┘
┘	│	┘	├	┌	┐
┐	│	┬	─	│	┼
└	┌	─	─	└	─
─	│	┘	─	└	┬
┬	┴	┌	─	┬	│
│	┐	│	┐	┘	│

Default rotation is counter-clockwise.
Center of coordinates = left top corner.
NB: TableFixture? uses 0-indexed tables, but in the acceptance test, the client uses more intuitive 1-index tables.

apse502.f04.rocketmania.fit.CalculateScore		
press	traverse	
check	score	0
enter	row	6
enter	column	5
press	rotate	
press	rotate	
press	traverse	
check	score	160

apse502.f04.rocketmania.fit.BoardTraverser					
│	│	┴	┐	┘	│
└	└	┼	┌	┘	─
┘	┌	└	├	┘	┘
┘	│	┘	├	┌	┐
┐	│	┬	─	│	┼
└	┌	─	─	┐	─
─	│	┘	─	└	┬
┬	┴	┌	─	┬	│
│	┐	│	┐	┘	│

Figure 5. Sample FIT test from the Rocket Mania acceptance suite.

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 4 shows the suite of acceptance tests and Figure 5 shows a sample test case for traversing the board with multiple rotations of a single fuse. Teams had two and a half weeks to master FIT and to complete the assignment.

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 5).

Notice, unlike in the Web-based Systems course user acceptance testing in this course was introduced later during the semester, when some of the working code was already in place. We have observed better results in code re-use and implementation of the Delegate design pattern (in contrast, several teams in senior Web-based Systems course delivered "fat" fixtures that provided all contained all required functionality). Also, in this assignment, students were utilizing several fixtures (in the example depicted by Fig.5, 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.

During informal discussions, students praised FIT and indicated that it was more useful for describing functional requirements than prose. Many 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.

8 Summary

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

FIT can be incorporated in any class regardless of the software development methodology. In both courses, students commented that concentrating more on tests (including 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, instructors can choose one of three strategies: 1) to provide a complete acceptance suite and use it as the main assignment specification; 2) to specify an incomplete acceptance suite and encourage students to extend it; 3) to assign the task of writing acceptance tests to teams themselves and then exchange the suites. Writing acceptance tests first demonstrates a healthy approach to software

development. It emphasizes the importance of testing and students have enough time during the course to grasp the essence of FIT and to benefit from using it. They also have enough time to reflect on their experiences.

FIT also stimulates communication between the customer and the team. This communication is more effective as it is less prone to the sins of traditional requirements: ambiguity, noise, multiple representations, and customer uncertainty.

The simplicity of the FIT framework and Fitnesse engine allows for a quick and easy setup. We have used it equally well from 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 adding 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 (see Fitnesse discussion forum at yahoo).

What is more, 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 [8].

All of the above indicates, at minimum, that FIT deserves to be considered by academia more closely. We encourage more educators to try it out. Future studies will help to validate its usefulness both in academic and industrial settings.

References

1. Acceptance Test. Online, last retrieved Jan 25, 2005: http://c2.com/cgi/wiki?AcceptanceTest
2. Perry, W. Effective Methods for Software Testing, 2/e, John Wiley & Sons: New York, NY, 2000.
3. Faught, D., Bach, J. Not Your Father's Test Automation: An Agile Approach to Test Automation. Sticky Minds. Online, last retrieved Jan 25, 2005:

   http://www.stickyminds.com/sitewide.asp?Function=WEEKLYCOLUMN&ObjectId=8392&ObjectType=ARTCOL&btntopic=artcol&tt=WEEKLYCOL_8392_title&tth=H&commstatus=expand&ci=3746#3746
4. Fitnesse. Online, last retrieved Jan 25, 2005: http://fitnesse.org
5. Cunningham, W. Fit: Framework for Integrated Test. Online, last retrieved Jan 25, 2005: http://fit.c2.com/

6. Shore, Jim. Introduction to Fit. Online, last retrieved Jan 25, 2005: http://fit.c2.com/wiki.cgi?IntroductionToFit
7. RocketMania. Online, last retrieved Jan 25, 2005: http://games.yahoo.com/games/downloads/rm.html
8. Chaos Report. The Standish Group, West Yarmouth, MA, 1995, 1997, 1999, 2001, 2003.