FitClipse: A Fit-Based Eclipse Plug-In for Executable Acceptance Test Driven Development

Chengyao Deng, Patrick Wilson, and Frank Maurer

University of Calgary
Department of Computer Science
2500 University Dr. NW
Calgary, Alberta T2N 1N4 Canada
`{cdeng,piwilson,maurer}@cpsc.ucalgary.ca`

Abstract. We conducted a survey on Executable Acceptance Test Driven Development (or: Story Test Driven Development). The results show that there is often a substantial delay between defining an acceptance test and its first successful pass. Therefore, it becomes important for teams to easily be able to distinguish between tasks that were never tackled before and tasks that were already completed but whose tests are now failing again. We then describe our FitClipse tool that extends Fit by maintaining a history of acceptance test results. Based on the history, FitClipse is able to generate reports that show when an acceptance test is suddenly failing again.

Keywords: Executable Acceptance Test-Driven Development (EATDD), executable acceptance test, Fit.

1 Introduction

In Extreme Programming, two sets of test techniques are used for double checking the performance of a system, unit testing and acceptance testing. [7] With unit testing, detailed tests from the developer's perspective are conducted to make sure all system components are working well. Acceptance testing is the process of customers testing the functionality of a system in order to determine whether the system meets the requirements. Acceptance tests are defined by or with the customers and are the concrete examples of system features. Recent literature on agile methods suggests that executable acceptance tests should be created for all stories and that a story should not be considered to be completed until all the acceptance tests are passing successfully. [3][10] Acceptance tests should be expressed in the customer language (i.e. customers should be able to understand what they mean) and should be executable (i.e. automated) and be included in the continuous integration process.

Executable Acceptance Test Driven-Development (EATDD), which is also known as Story Test-Driven Development (STDD) or Customer Test-Driven Development, is an extension of Test-Driven Development (TDD). While TDD focuses on unit tests to ensure the system is performing correctly from a developer's perspective, EATDD starts from business-facing tests to help developers better understand the requirements,

to ensure that the system meets those requirements and to express development progress in a language that is understandable to the customers. [11]

From the customer's perspective, EATDD provides the customer with an "executable and readable contract that the programmers have to obey" if they want to declare that the system meets the given requirements. [12] Observing acceptance tests also gives the customers more confidence in the functionality of the system. From the perspective of programmers, EATDD helps the programmers to make sure they are delivering what the customers want. In addition, the results help the team to understand if they are on track with the expected development progress. Further, as EATDD propagates automated acceptance test, these tests can play the role of regression tests in later development.

This paper is organized as follows: section 2 discusses a survey and the motivations for building such a tool; section 3 presents the related work and Eclipse plug-ins based on Fit; section 4 describes the overall design of FitClipse; section 5 talks about how FitClipse works for EATDD; section 6 demonstrates our initial evaluation of FitClipse.

2   Survey Results and Motivation

We conducted a survey to find out how EATDD is being used in industry by sending questionnaires to mailing lists and discussion groups of Agile communities. The comprehensive findings of this study will be published in the future. One specific part of that study is relevant for this paper: We asked about the time frame between defining an acceptance test and its fist successful passing. The findings of this questionnaire are a core motivation underlying the development of FitClipse.
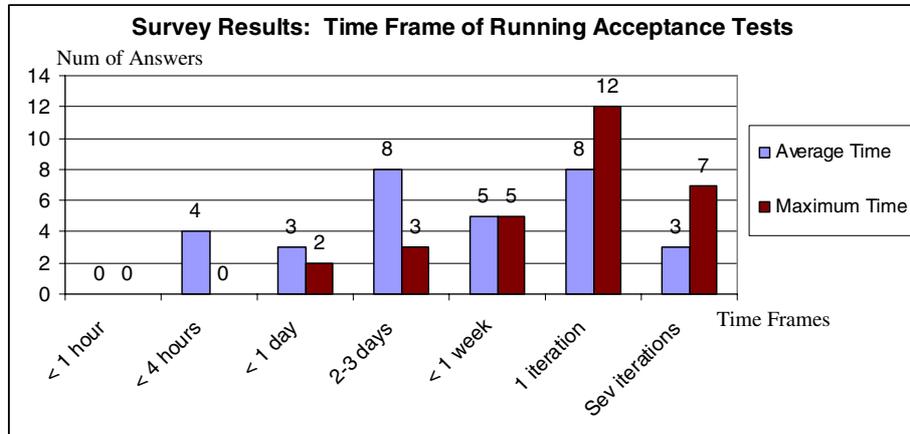
2.1   Timeframe of EATDD

A major difference between TDD using unit tests and EATDD is the timeframe between the definition of a test and its first successful pass. Usually, in TDD the expectation is that all unit tests pass all the time and that it only takes a few minutes between defining a new test and making it pass [8]. As a result, any failed test is seen as a problem that needs to be resolved immediately. Unit tests cover very fine grained details which makes this expectation reasonable in a TDD context.

Acceptance tests, on the other hand, cover larger pieces of functionality. Therefore, we expected that it may often take developers several hours or days, sometimes even more than one iteration, to make them pass.

For validating our hypothesis, we conducted a survey by sending a questionnaire to email groups of Agile Communities (such as the Yahoo agile-usability group and the Yahoo agile-testing group etc.). One goal of the survey was to find out the timeframe between the definition of an acceptance test and making it pass successfully. We were expecting the following results:

- The *average* timeframe between defining one acceptance test and making it pass successfully, following EATDD, is more than 4 hours (half a day).
- The *maximum* timeframe between defining one acceptance test and making it pass successfully, following EATDD, may be the majority of an iteration or even more than one iteration.

Survey Results: Time Frame of Running Acceptance Tests

Num of Answers

[Bar chart showing survey results across time frames (< 1 hour, < 4 hours, < 1 day, 2-3 days, < 1 week, 1 iteration, Sev iterations) with Average Time (light bars) and Maximum Time (dark bars). Values: < 1 hour: 0, 0; < 4 hours: 4, 0; < 1 day: 3, 2; 2-3 days: 8, 3; < 1 week: 5, 5; 1 iteration: 8, 12; Sev iterations: 3, 7]

Fig. 1. This Chart shows the survey results of the time frame between the definition of an acceptance and making it pass successfully

Overall, we received 33 responses, among which 31 were valid. Fig. 1 shows the detailed findings of the survey according to the above expectations.

The result of the survey strongly supports our first expectation. About 87.1% (27/31) of the participants reported the average timeframe to be more than 4 hours for defining an acceptance test and making it pass and the number increased to 100% when they reported the maximum time.

Our second expectation is also supported by the survey result. 41.4% (12/29, two participants did not answer) of the participants spent most of an iteration to finish one acceptance test, and about 24.1% (7/29) of the participants reported the time frame to be several iterations. One of the participants even spent several months working on making a single acceptance test pass.

Therefore, both of our expectations were substantiated by the evidence gathered in this survey. We can also draw the conclusion that the time frame between the definition of an acceptance test and its first successful pass is much longer than that of unit test.

2.2 Motivation of FitClipse

Due to the substantial delay between the definition and the first successful pass of an acceptance test, a development team can NOT expect that all acceptance tests pass all the time. A failing acceptance test can actually mean one of two things:

- The development team has not yet finished working on the story with the failing acceptance test (including the developer has not even started working on it).
- The test has passed in the past and is suddenly failing – i.e. a change to the system has triggered unwanted side effects and the team has lost some of the existing functionalities.

The first case is simply a part of the normal test-driven development process: It is expected that a test that has never passed before should fail if no change has been made to the system code. The later case should be raising flags and should be highlighted in progress reports to the team. Otherwise the users have to rely on their recollection of past test results to determine the meaning of a failing test. For anything but very small projects, this recollection will not be reliable.

FitClipse raises a flag for the condition that a test that is failing but was passing in the past. It identifies this condition by storing the results of previous test executions and is, thus, able to distinguish these two cases and splits up the "failed" state of Fit into two states: "still failing" and "now failing after passing earlier".

3   Related Work

There are several open source frameworks and tools that support EATDD, with Fit [4], FitLibrary and FitNesse [5] being three of the most relevant to our work.

Fit is a framework for integrated testing. "It is well suited to testing from a business perspective, using tables to represent tests and automatically reporting the results of those tests."[9] Fixtures, made by the programmers to execute the business logic tables, map the table contents to calls into the software system. Test results are displayed using three different colors for different states of the test: green for passing tests; yellow for tests that can not be executed and red for failing tests. FitLibrary is a collection of extensions for the fixtures in Fit. Other than test styles that Fit provides, it also supports testing grids and images. FitNesse is a Wiki front-end testing tool which supports team collaboration for creating and editing acceptance tests. FitNesse uses the Fit framework to enable running acceptance tests via a web browser. It also integrates FitLibrary fixtures for writing and running acceptance tests.

Our FitClipse tool is an Eclipse plug-in that uses the Fit framework for writing and running the acceptance tests. There are several other Eclipse plug-ins which also use Fit or FitNesse, including FitRunner [6], conFIT [2] and AutAT [1]. FitRunner contributes to Eclipse a Fit launch configuration that enables people to run automated acceptance tests. ConFIT uses a FitNesse server, which can run either locally or remotely, to perform acceptance tests. AutAT enables non-technical users to write and execute automated acceptance tests for web applications using a user-friendly graphical editor. [1]

Compared to the above tools and in addition to running acceptance tests with the Fit frame work, FitClipse extends the Fit tests result schema with historical result information for the users. In FitClipse, instead of one single test failure state, two kinds of acceptance test failure states are distinguished automatically: unimplemented failure and regression failure.

4   FitClipse

FitClipse [13] is an Eclipse plug-in supporting the creation, modification and execution of acceptance tests using the FIT/FitNesse frame work.

FitClipse works as a client side application and communicates with a Wiki repository, which works as the server. The repository has been implemented with FitNesse. The FitClipse tool consists of (multiple) FitClipse clients for editing and running acceptance tests, the Wiki repository for storing acceptance test definitions and the Database for storing the test execution history (See Fig. 2). Using FitClipse, acceptance tests are written, on the client side, in the form of executable tables with Wiki syntax and then saved on the server side as Wiki pages. FitClipse uses the Fit engine to run the acceptance tests.
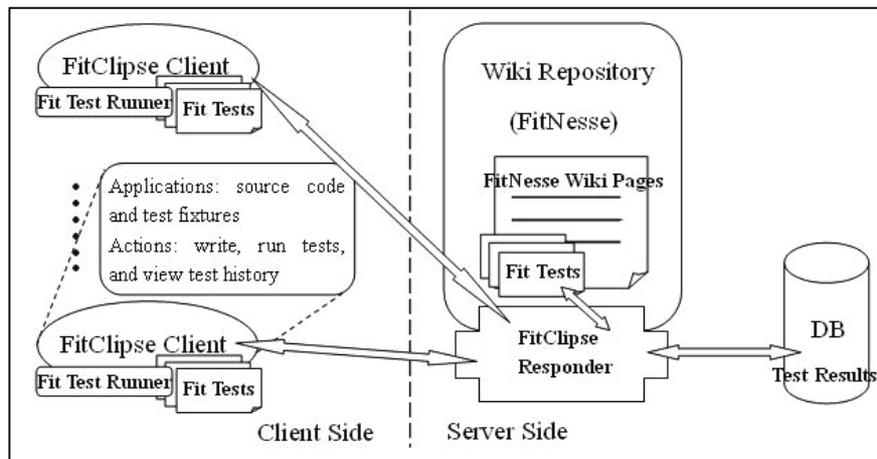
Fig. 2. Overview of FitClipse framework with FitNesse as the backend. We extend FitNesse server by adding a FitClipse Responder to handle the requests from FitClipse clients and to talk with the database for saving and retrieving the test results.

In order to distinguish two test failure states, FitClipse, coupled with a Wiki repository server, stores the results of each test run and can retrieve the result histories for each test and each test suite. The algorithm for distinguishing two test result failures is as follows:

```
for (each test t){
    t.run();
    PersistTestResult (t.result);
    if (t.isFailing){
       getResultHistory(t);
       If (hasPassedBefore(t)){
          displayRegressionFeature();
       }else
          displayUnimplementedFailure(t);
}}}
```

FitClipse splits up the test failure state in Fit or FitNesse into two: Unimplemented failure and Regression failure. Table 1 shows the four test result states in FitClipse comparing them to the three states of Fit or FitNesse.

Table 1. Comparison of test result states of FitClipse and Fit or FitNesse

Test Result States	Fit or FitNesse	FitClipse
Failure (the tests fail)	Color Red	Regression Failure – failure as a result of a recent change losing previously working functionality (color red)
		Unimplemented Feature – not really a failure as it might simply mean that the development team hasn't started to work on this feature (color orange)
Passing (the tests pass)	Color Green	test page with green bar – no difference to Fit/FitNesse (color green)
Exception (the tests can not be executed)	Color Yellow	test page with yellow bar – no difference to Fit/FitNesse (color yellow)

5   FitClipse and EATDD

FitClipse provides the following core functionalities for EATDD:

1) *Create and modify Acceptance Tests*: In the FitClipse environment (as shown in Fig. 3), customer representatives, testers and developers collaborate on creating acceptance tests for each story. Users can create, delete and restructure acceptance tests in FitClipse.

2) *Creating Fixtures*: The programmers create fixtures that translate Fit tables into calls to the system under development. Based on a given acceptance test, FitClipse can generate the fixture code stubs automatically. Fig. 3 shows sample Fit tests and corresponding fixture code in FitClipse.
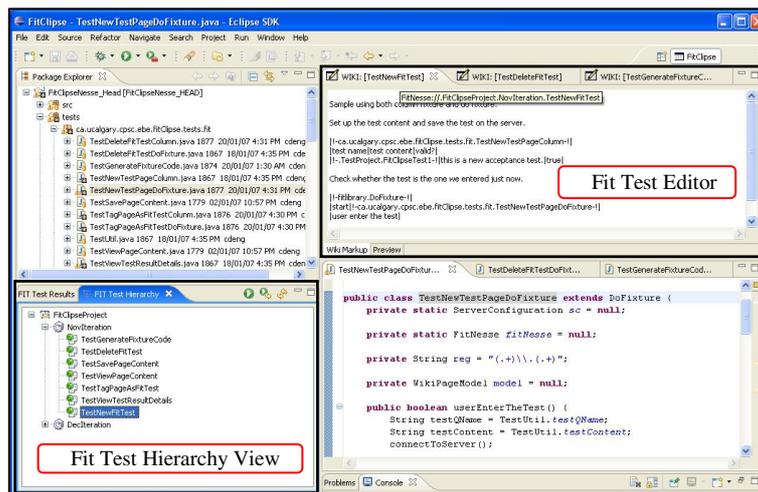
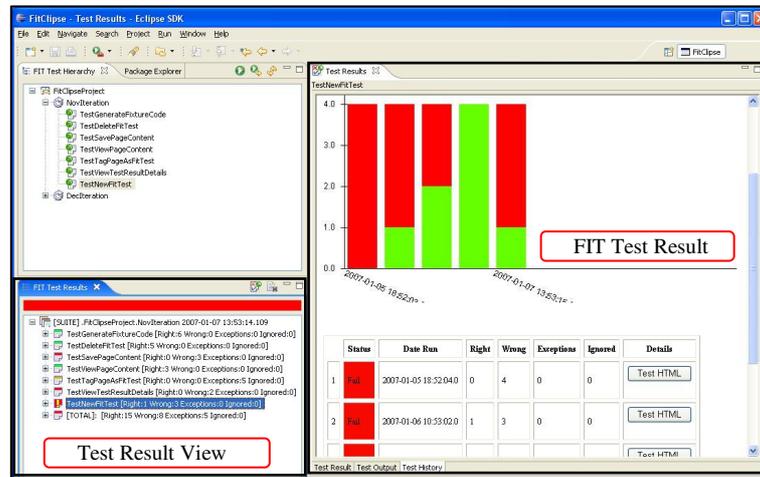**Fig. 3.** Fit tests and fixture code in FitClipse environment

Fig. 4. Test result states and test result history for an acceptance test

3) *Implementation*: In this step, unit test-driven development is utilized in conjunction with EATDD. Programmers follow TDD to implement features of the system. After several unit tests are passing, one acceptance test will pass, too. All through the implementation of acceptance tests, FitClipse provides two kinds of test failure states and maintains the test result history. Fig. 4 shows all test states and a sample test result history in FitClipse.

6   Initial Evaluation

We ran an initial self-evaluation on FitClipse by using it for two iterations. The evaluation lasted for 6 weeks and had several findings.

We followed EATDD process in our development process. In all, we spent about 150 programming hours and created 14 acceptance tests with 40 assertions.

Our first observation confirms our expectations. The distinction between two test failure states was helpful when the number of acceptance tests increased. When we broke the system by adding new code, the second failure state warned us at once by showing the special flag. We did not have to trace in a test history record or rely on our memory to recognize which test was passing before and broken by ourselves.

Second, the test result history provided helpful information for us to understand the development progress. In FitClipse, we can generate a test result history chart for a suite which includes all the acceptance tests in the iteration. From the number of passing and failing acceptance test we could see how our development was progressing.

Even though we only have limited time to evaluate FitClipse, we find that it was worth the effort as the distinction between two test failure states is useful. We believe that if we had spent a longer time for the evaluation with more acceptance tests, we would find the tool even more helpful.

To address the self-confirmation bias of the initial self evaluation, we will conduct a controlled experiment using outsiders in February and March 2007.

7   Conclusion and Future Work

This paper presents FitClipse, a Fit-based tool for automated acceptance testing and a self-evaluation of the tool.

Existing tools are limited in supporting Acceptance Test Driven Development as they do not provide enough information to distinguish two different kinds of test failures. FitClipse distinguishes these failure states by maintaining a test result history on the server, which is valuable for analyzing the existing progress and making improvements.

From the self-evaluation, we can see that FitClipse can provide useful support for EATDD. However, this self-evaluation is limited in time and the number of acceptance tests. Therefore, the next research step is to conduct a more formal evaluation of the approach to assess if FitClipse as a whole is useful for development teams to practice Executable Acceptance Test Driven Development. In the future, FitClipse will also provide a WYGIWYS editor for supporting the users to edit the Fit test documents.

References

1. Schwarz, C., Skytteren, S.K., Øvstetun, T.M.: AutAT – An Eclipse Plugin for Automatic Acceptance Testing of Web applications. OOPSLA'05. October 16–20, 2005, San Diego, California, USA (2005) ACM 1-59593-193-7/05/0010 (See also: http://boss.bekk.no/autat/)
2. conFIT: A FitNesse for Eclipse Plugin (http://www.bandxi.com/fitnesse/)
3. Extreme Programmin: Acceptance Tests, (http://www.extremeprogramming.org/rules/functionaltests.html)
4. Fit: Framework for Integrated Test (http://fit.c2.com/)
5. FitNesse Web site (http://fitnesse.org/)
6. FitRunner: an Eclipse plug-in for Fit (http://fitrunner.sourceforge.net)
7. Beck, K.: Extreme Programming Explained: Embrace Change. Addison Wesley, Boston (2000)
8. Beck, K.: Test-Driven Development: By example, p. 11. Addison –Wesley, London (2003)
9. Mugridge, R., Cunningham, W.: Fit for Developing Software: Framework for Integrated Tests, p. 1. Prentice Hall, Englewood Cliffs (2005)
10. Miller, R.W., Collins, C.T.: Acceptance Testing, 2001 XP Universe Conference, Raleigh, NC, USA (July 23–25, 2001)
11. Story Test Driven Development (http://sangam.sourceforge.net/StoryTestDrivenDevelopment)
12. Tracy Reppert, Do't Just Break Software, Make Software, Better Software Magazine (July/August 2004) available on line: http://industriallogic.com/papers/storytest.pdf
13. University of Calgary, EBE website: FitClipse, (http://ebe.cpsc.ucalgary.ca/ebe/Wiki.jsp?page=.FitClipse)