Design and Implementation of a Tool for Executable Acceptance Test Driven Development

Azarm Mazandarani, Mohamad Javad Rostami, and Ali Mohammad Norouzzadeh, *Member, IACSIT*

***Abstract*—This paper introduces AcceptSoftware which is a tool to easily create and run client readable acceptance tests, and describes how it can be used to allow a simple but powerful acceptance-test driven software development. We then describe our AcceptSoftware tool that extends EasyAccept by maintaining a history of acceptance test results. Based on the history, AcceptSoftware is able to generate reports that show when an acceptance test is suddenly failing again.**

***Index Terms*—Software testing, acceptance test, ATDD, test-driven development.**

I. Introduction

Acceptance testing is an important aspect of software development. Acceptance tests (sometimes referred as story tests in agile teams) are high level tests of business operations. They are not meant to test internals or technical elements of the code, but rather are used to ensure that software meets business goals. Executable (i.e. automated) acceptance tests can be used as a measure of project progress.

As the software system becomes more complex, analysts spend more time on requirements specifications. A solution is to repeat the development cycle in small incremental iterations, as recommended by agile methods [1]. One of the biggest contributions of agile methodologies is the concept of test-driven development (TDD). In TDD, the tests are written before writing the actual code. The tests can be used to evaluate the development progress by measuring the number of passing or failing tests and to perform continuous regression testing, which can help maintain high software quality by notifying the developers of software defects as soon as the code is changed.

Automated acceptance tests [2] are used in TDD which is called Executable Acceptance Test Driven Development (EATDD). It is also known as Story Test Driven Development or Customer Test Driven Development. Acceptance tests for a feature should be written first by the customer with the help of the development team, before the application code is implemented. The tests represent system requirements and specifications. Then, the development team will work on implementation with guidance of the acceptance tests. The implementation is completed when all the corresponding acceptance tests are passed.

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 [3].

There is often a substantial delay between defining an acceptance test and its first successful pass [4]. 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. This is achieved by using AcceptSoftware.

This paper introduces AcceptSoftware which is a tool to easily create and run client readable acceptance tests, and describes how it can be used to allow a simple but powerful acceptance-test driven software development. We then describe our AcceptSoftware tool that extends EasyAccept [5], [6] by maintaining a history of acceptance test results. Based on the history, AcceptSoftware is able to generate reports that show when an acceptance test is suddenly failing again.

The rest of this paper is organized as follows. We review the related work in Section II. We then review EasyAccept and present our motivation to improve it in Section III, and also we introduce AcceptSoftware. We discuss its implementation in Section IV. Section V contains our evaluations. Finally, Section V concludes the paper.

II. Related Work

In this section, we review existing researches and tools related to EATDD. We divide them into three categories as the following sub-sections.

A. Table-Based Frameworks

There are several open-source frameworks and tools that support EATDD. Table-driven tests are best suited to express business rule examples in input-output pairs that can be linked to the business logic algorithmically. On the other hand, sequential command-driven tests are suited to express the business logic workflow. It is well suited to testing from a business perspective, using tables to represent tests and automatically reporting the results of those tests.

Examples of tools in this category include Fit [7], Fitness [8], and Selenium [9]. The most widely known tool for acceptance testing is Fit (Framework for Integrated Testing). Fit requires developers to design individual fixture classes

Manuscript received October 1, 2011; revised December 15, 2011.

Azarm Mazandarani is with the Department of Engineering, Azad University, Kordkouy Branch, Kordkouy, Iran. (e-mail: azarm.mazandarani@gmail.com)

Mohamad Javad Rostami is with the Department of Computer Engineering, Bahonar University, Kerman, Iran (e-mail: mjrostamy@yahoo.com).

Ali Mohammad Norouzzadeh is with Guilan Science and Technology Park, Rasht, Iran (e-mail: ahmadi.abbas64@yahoo.com).

with hookup code for every type of table used in the tests and cope with data being referenced across tables.

B. Text-Based Frameworks

Although table-based frameworks might be the mainstream right now, they are not the only class of frameworks suitable for acceptance testing. Not everyone likes authoring tests as tables. The text written into the cells of test tables is often close to written English, but the table structure brings with it a degree of syntax.

Text-based tests are written as simple texts using a text editor. These kinds of tests are useful to represent work flows [10]. Examples of tools in this category include Exactor [11], Text Test [12], Easy Accept, JAccept [13]. Exactor uses textual scripts, JAccept is based on a graphical editor and XML test files, and Text Test tests programs with command-line textual input and output. They are suited to express the business logic workflow.

C. Scripting Language-Based Frameworks

There is another category of acceptance-testing tools that can offer a great deal of power through flexibility and friendliness of a scripting language. A good example of this category of tools is Systir [14] which makes use of the Ruby scripting language's syntax for building reasonably-good custom domain-specific languages.

III. EasyAccept and Motivation to Improve

AcceptSoftware tool extends EasyAccept by maintaining a history of acceptance test results. EasyAccept is an open-source tool that can be found at [6]. It takes acceptance tests enclosing business rules and a Façade to access the software under development, and checks if the outputs of the software's execution match expected results from the tests. Driven by EasyAccept runs, software can be constructed with focus, control and correctness, since the acceptance tests also serve as automated regression tests.

In short, EasyAccept is a script interpreter and runner. It takes tests enclosed in one or more text files and a Façade to the program that will be tested. Accessing the program through Façade methods that match user-created script commands, EasyAccept runs the entire suite of tests and evaluates actual and expected outputs or behaviors of the program under test. In a test report, the tool shows divergences between actual and expected results, or a single message indicating all tests were run correctly.

The acceptance tests are written in text files with user-created commands close to natural language. EasyAccept provides some built-in commands which are combined with such customized user-created commands specific for each application to create the tests.

The overhead of getting started with EasyAccept is practically zero, and it requires minimal additional work on the part of the developers. They only need to provide a Façade to the program to be tested containing methods whose signatures match the user-created commands. A single Façade that exposes the program's business logic helps separate business and user interface concerns, and may even already exist in programs not created with an ATDD approach, since this separation is an advocated architectural best practice. Other textual testing tools use various approaches, none of which involves the use of a single Façade.

A. Motivation to Improve

A major difference between UTDD and EATDD is the timeframe between the definition of a test and its first successful pass. In UTDD, 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 [15]. As a result, any failed test is considered as a problem that needs to be resolved immediately. Unit tests cover very fine grained details which make this expectation reasonable in a TDD context.

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

1) Non-implemented Feature: 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).
2) . Regression Failure: The test has passed in the past and is suddenly failing, i.e., a change to the system has triggered undesired side effects and the team has lost some of the existing functionalities.

Keeping history of number of passed and failed acceptance tests of a project helps the development team understand the development progress. From such statistics, the development team can grasp the speed of their development and where they are in the development process.

AcceptSoftware has the functionality of showing the test result history. Test result history is kept in the database. To show the test result history, a chart showing the test running date and result details are provided.

Changes are often made to acceptance tests. Most people make changes to acceptance tests many times a day when they come up with new ideas. Acceptance tests which were changed before might need to be reversed back to a previous version. However, only keeping the version information is not sufficient enough. Sometimes the developers or tests make improper changes and keep adding changes to the tests for a period of time. Afterwards, when people discover the mistake, provided only a version number and a date, it is very hard for them to decide which version of the test is useful. It will be very helpful if the test result information can be kept with the corresponding versions of the test. By viewing the test results, people can easily identify the test that is performing as expected. AcceptSoftware achieves this goal by keeping test result record after each test run. In addition, identifying the regression failure of acceptance tests requires keeping history of the tests to identify the last version of successful tests.

Acceptance tests can be divided into the following categories.

1) Tests containing lots of information and formulas. It is efficient to represent such tests using tables.
2) Tests containing job rules. It is efficient to represent such tests using texts.

None of the existing EATDD tools supports both the above categories of tests. In addition, none of the existing EATDD tools keeps history of tests. We developed AcceptSoftware that adds these two features into EasyAccept.

B. The AcceptSoftware Tool

Fig. 1 demonstrates the test framework which is used in AcceptSoftware. AcceptSoftwrae extends EasyAccept by maintaining a history of acceptance test results. A class called Façade is used to call procedures of the under-test program. All commands in test scripts must be compatible to Façade's methods. Façade helps the developer in the future when the developer implements a user interface.

AcceptSoftware contains the same internal commands used in EasyAccept except for an internal command called expectable. We have extended this command in AcceptSoftware providing the possibility in AcceptSoftware to read a data table from a database (including Oracle, Access, MySQL, and SQL-Server databases) and then use the data to test the program.

AcceptSoftware keeps a complete history of test cases and test results in a database. This makes it possible to avoid redundant test cases when the software is under development. For example, let us consider a program which is recently developed. The programmer defines a large number of test cases and starts testing the program. After this stage of testing, a number of faults are detected in the program. Then, the programmer tries to change the program to remove the faults. Then, the programmer does another phase of testing. If the programmer uses an existing tool for testing, he/she has to repeat all the test cases again. Using AcceptSofware, the programmer needs to repeat only the following two sets of test cases.

1) The test cases which failed at the previous stage of testing.
2) The test cases that depend on the new changes in the program.

Since AcceptSoftware decides on the new test cases according to the results of the previous test cases, it also considers the relation between test cases to improve test case reduction. AcceptSoftware considers the following kind of relation between test cases *a* and *b*.

1) If *a* fails, then *b* may fail: This implies that we have to include test case *b* in the new stage of testing.
2) If *a* fails, then *b* passes: This implies that we have to exclude test case *b* in the new stage of testing, only if *a* fails.

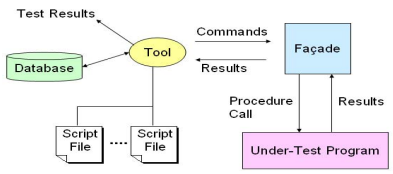

Fig. 1. Overview of AcceptSoftware

3) If *a* passes, then *b* may fail: This implies that we have to include test case *b* in the new stage of testing.
4) If *a* passes, then *b* passes: This implies that we have to exclude test case *b* in the new stage of testing, only if *a* passes.

These are the relations leading to a certain decision in either including or excluding test case *b*. Other kinds of relations do not lead to a certain decision and can not be used.

IV. Implementation of AcceptSoftware

In this section, we describe our implementation of AcceptSoftware.

A. Class AcceptSoftware

This class is the core of AcceptSoftware that manages operations such as detecting the Façade of the under-test program, detecting test script files, and doing test operations for each script file.

B. Class AcceptRunner

This class tests the program using a procedure called runnScript() considering the script file. The test operation is done using a method in the Script class.

C. Class Script

This class contains a method called runn() that runs the script on the under-test program and reports the result. Another method in this class called execute() helps in execution of the scripts. To properly perform the tests using the script file, this class parses the script file and associatively accesses Façade.

D. Class ParsedLineReader

Using method getParsedLine() in this class, the script files is parsed line by line and keywords are searched.

E. Tokens

Tokens are the keywords used to write the scripts. The tokens defined in AcceptSoftware include: echo, expect, expectdifferent, expecterror, expectwithin, equalfiles, quit, stringdeli, iter, stacktrace, executescript, threadpool, repeat, expectable.

F. Class ExpectTableProcessor

This class searches the filename or the id of the database in the script file. This operation is successful only when the tool reads keyword "expectTable" before the name of the database. Then, it connects to the database and reads the data table. It creates a new script file containing the data and executes this file. In this way, the data stored in a database can be used for testing a program.

G. Class DatabaseHandler

This class handles detection of database type, connecting to database, reading data from database, and creating the script file from it.

H. Database Implementation to Keep Test History

One of the advantages of AcceptSoftware over EasyAccept is the capability of storing data and statistics which are related

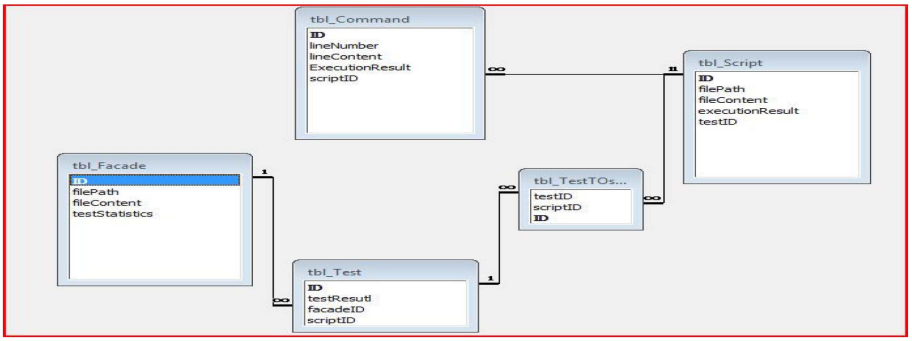

Fig. 2. Defined tables and their relationship.

to different executions of the under-test program. To achieve this feature, we implemented a database to log all the events and statistics related to execution of the program.

We define the following five tables (Fig. 2) in the database for keeping test history.

1) tbl_Test
2) tbl_Script
3) tbl_Command
4) tbl_Facade
5) tbl_TestTOscript

A record is stored in a table called tbl_Test for each under-test façade. A record is stored in a table called tbl_Script for each script file which is tested on the façade. A script file contains a number of command lines. Each command line is stored in a table called tbl_Command as a record.

Applying a command line during testing, the test result is updated in tbl_Command. This process continues until all the command lines are executed. Then, the result of executing the entire script file is updated in tbl_Script. Finally when the façade is tested by script files, the total result of this version of tests is updated in a table called tbl_Test.

A façade may be tested multiple times in the database. In this case, only one record is inserted in tbl_Facade whereas multiple records are inserted in tbl_Test. This feature avoids data redundancy and makes it easier to report a façade.

V. Performance Evaluation

In this section, we evaluate AcceptSoftware compared with EasyAccept and then review the results. We define two scenarios defined in Table I and II. In Scenario I, we evaluate AcceptSoftware and EasyAccept in testing an accounting program while the parameters are fixed. In Scenario II, we change initial number of test cases in different experiments.

TABLE I: Evaluation Parameters in Scenario I

Parameter	Value
Program under test	An accounting program written in C++
Initial number of test cases	1147
Number of test stages	8
Number of changes in program per test stages	12

A test experiment contains a number of stages. Initial number of test cases is the total number of test cases which can be included. They are all included in the first stage of testing. As we move to the next stage, the same test cases are included in testing when we use EasyAccept. As we move to the next stage, fewer test cases are included in testing when we use AcceptSoftware.

TABLE II: Evaluation Parameters in Scenario II

Parameter	Value
Program under test	An accounting program written in C++
Initial number of test cases	variable from 100 to 10000
Number of test stages	8
Number of changes in program per test stages	proportional to number of test cases

A. Numerical Results

Fig. 3 shows number of test cases which have to be considered in stages of testing in Scenario I. Since EasyAccept does not keep history of tests, it has to reconsider all the test cases in the next stages. In contrast, AcceptSoftware reduces number of test cases averagely by 55 percent whenever it moves to next stage. In the 8th stage, AcceptSoftware needs only 2 test cases.

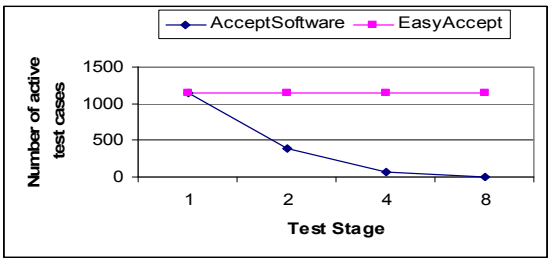

Fig. 3. Number of active test cases versus test stage (Scenario I)

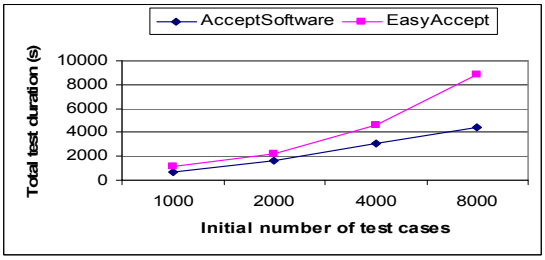

Fig. 4. Total test duration versus initial number of test cases (Scenario II)

Fig. 4 shows total time required for testing versus initial number of test cases in Scenario II. Since Easy Accept has to reconsider all test cases in the next stages, doubling number of test cases leads to doubling test duration. This feature reduces scalability of Easy Accept. In contrast, when using AcceptSoftware, doubling number of test cases leads to averagely 73 percent increase in test duration. This is because of the fact that the more initial test cases we have the more test cases AcceptSoftware is able to exclude in the next stage.

VI. Conclusion

This paper presents AcceptSoftware which is an EasyAccept-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. AcceptSoftware distinguishes these failure states by maintaining a test result history on the server, which is valuable for analyzing the existing progress and making improvements. Table III compares AcceptSoftware with the existing open-source tools of acceptance testing.

As a tool supporting agile methodology, it will be helpful to integrate this work with other practices in Agile. For instance, acceptance tests can be used in conjunction with story card management to provide more meaningful reports for the customers.

The work presented in this paper is a preliminary step in constructing an effective tool for supporting EATDD in Agile software development environment. There is still a lot of room in this research area for future work.

From the self-evaluation, we can see that AcceptSoftware 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 AcceptSoftware as a whole is useful for development teams to practice executable acceptance test driven development.

Another idea for future work involving AcceptSoftware is a comparison to other ATDD approaches, particularly those that use different formats of acceptance tests such as FiT tables. Such a comparison would allow us to abstract away which ATDD patterns and techniques are tool-dependent and which are general, improving the state-of-the-art of acceptance testing.

TABLE III: Comparison of Software Testing Tools

Tool		TextTest	Exactor	EasyAccept	Selenium	FIT	AcceptSoftware
Acceptance Testing Criteria	Edit/Run	*	*	*	*	*	*
	Supporting the text format	*	*	*	-	-	*
	Supporting the HTML format	-	-	-	*	*	*
	Supporting the Excel format	-	-	-	-	*	*
	SQL, Oracle, XML format	-	-	-	-	-	*
Test Result Criteria	Detection of regression errors and non-implemented features	-	-	-	-	-	*
	Presenting test result history	-	-	-	-	-	*
Other Criteria	Open source	*	*	*	*	*	*
	Being user friendly	*	*	*	*	*	*
	Containing a Façade	-	-	*	-	-	*

References

[1] Calgary Agile Method User Group home, Available: http://www.agilenetwork.ca/camug/.

[2] A. Neto, J. P. Sauvé, and A. Dantas, "Patterns for Scripted Acceptance Test-Driven Development," in *Proc. the 12th European Conference on Pattern Languages of Programs*, Irsee Monastery, Germany, 2007, pp. A4.1-A4.13.

[3] L. Koskela, *Test Driven: practical TDD and acceptance TDD for Java developers*, USA: Manning Publications, 2007.

[4] L. Crispin, T. House, and C. Wade, "The Need for Speed: Automating Acceptance Testing in an Extreme Programming Environment," in *Proc. Second Int'l Conf. eXtreme Programming and Flexible Processes in Software Eng*, 2001, pp.96-104.

[5] J. P. Sauvé, A. Neto, and W. Cirne, "EasyAccept: a tool to easily create, run and drive development with automated acceptance tests," in *Proc. the 2006 international workshop on Automation of software test*, 2006.

[6] EasyAccept, Available: http://easyaccept.org.

[7] FiT, Available: http://agile.csc.ncsu.edu/SEMaterials/tutorials/fit/.

[8] R. Mugridge, and W. Cunningham, *Fit for Developing Software: Framework For Integrated Tests*, USA: Prentice Hall, 2005.

[9] Selenium homepage on OpenQA, Available: http://www.openqa.org/selenium/.

[10] J. Andersson, G. Bache, P. Sutton, "XP with Acceptance-Test Driven Development: A rewrite project for a resource optimization system," in *Proc. the 4th International Conference on Extreme Programming*, 2003.

[11] Exactor, Available: http://exactor.sourceforge.net/.

[12] TextTest, Available: http://texttest.carmen.se/.

[13] Jaccept, Available: http://maven.agilos.org/sites/jaccept/released/.

[14] Systir, Available: http://systir.rubyforge.org/.

[15] K. Beck and C. Andres, *Extreme Programming Explained*, 2nd ed. Boston, USA: Addison-Wesley, 2005.

Azarm Mazandarani was born in 1979 in Iran. Ms. Mazandarani received her B.Engr. degree from University of Science And Technology (Behshahr, Iran) in 2002 and her M.S. degree at Emam-Hosein University (Tehran, Iran) in 2008 in computer engineering. Since 2006, she is working as a lecturer at Azad University, Gorgan branch. She is interested in the area of software engineering.

Mohammad Javad Rostami was born in 1978 in Iran. He received his B.Sc in computer engineering from Bahonar University (Kerman, Iran) in 2001 and M.Sc in computer engineering from Amirkabir University of Technology (Tehran, Iran) in 2005. He has been a faculty member of Bahonar University since 2006. His main research interests include diverse routing and QoS routing algorithms, wireless sensor networks, and heuristic network algorithms. Mr. Rostami is a member of International Association of Computer Science and Information Technology.

Ali Mohammad Norouzzadeh was born in 1984 in Iran. Mr. Norouzzadeh received his B.Engr. degree from Azad University, Lahijan branch (Lahijan, Iran) in 1996 and his M.S. degree from Azad University, Gazvin branch (Qazvin, Iran) in computer engineering in 2011. Since 2008, he is working as a lecturer at Azad University, Gazvin branch. He is interested in the area of wireless networks and network engineering.