The Application of Multi-modal Test Execution Using Fitclipse

Shelly Park, Frank Maurer
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
{parksh, maurer}@cpsc.ucalgary.ca

Abstract

Multi-modal test execution allows execution of the same test against various layers and components of a software system. This paper presents a method that effectively encodes one-to-many test definition and test execution relationship of multi-modal functional tests without creating large test maintenance overhead. Our approach extends the Fit table specification structure by multi-modal fixtures in Fitclipse and presents the results of test execution in a way that can help debugging and progress reporting. We analyze the application of multi-modal test execution and the potential benefits of using multi-modal test execution in a multi-functional team.

1. Introduction

In Executable acceptance test driven development, the requirements are presented in form of tests. The benefit of executable acceptance testing is to reduce the ambiguous requirements specifications by enforcing the customer to specify the requirements with testable data and examples. The main difference of executable acceptance tests with other tests is that it defines the *functional requirements* criteria for a finished product from business perspective by the customer. In an EATDD environment, the acceptance tests are the main communication channel for communicating project progress, requirements and quality. All stakeholders of the software project including the development team are involved with varying degrees in specifying, testing, implementing or communicating through acceptance tests. Acceptance tests are not just for customers, but an important communication tool for all stakeholders – users, customers, analysts, developers and QAs – in the development project. In addition, agile methodologies consider automation of acceptance tests to be an essential part of the development [1,7], because many of the testing tasks are highly repetitive and postponing all acceptance tests to the end is "a bottleneck before the product delivery", which could have "overwhelming overhead for each delivery" [7] – and it often has as illustrated by many schedule overheads due to late integration and acceptance testing.

It is important to note that research into executable acceptance testing is not just about testing. Unlike the authors of the other types of tests, the author or the owner of executable acceptance tests is not the QAs. The specifications are supposed to be written by customers who often have limited software engineering background. The purpose of the executable acceptance tests is to communicate the business perspective of the software requirements and business values as much as it is to communicate the quality of the software. The challenge with executable specification is the format and the interpretation of the test results must be easily understood by the non-developers and non-testers, but also be able to convey the details of the development project from business perspective.

There are several tools and languages for representing and testing for workflow process and system requirements. The "record-and-playback" type of tools such as WATIR [2] or Selenium [3] work on the user interface layer. Swimlane is used for process flow diagrams, which is more popular among business process analysts to communicate the requirements. Finally, agile practitioners use Fit [4] or tools based on Fit such as FitNesse [5] and FitLibrary [6]. In Fit, the test definition is specified in form of tables, which is called "Fit tables". The developers write "fixtures" that map the test definition to the software program code. Once the Fit tables are combined with the fixtures, anyone can execute and read the test results.

We argue that the next generation of functional testing tools needs to be multi-modal: tests need to be expressible in multiple formats to satisfy the requirements from different stakeholder groups and need to be executable against different layers of the software system. We call the first *multi-modal test definition* and the second *multi-modal test execution*. This paper deals specifically with *multi-modal test*

execution and its applications in executable acceptance test driven development (EATDD) environment. Functional requirements do not necessarily translate easily to a specific part of the code, but rather functional requirements are combination of different parts of the code. Identifying these code easily and be able to map a functional feature to the location in the code is necessary for quality assurance activity. We have implemented multi-modal test execution in an executable acceptance testing tool called Fitclipse and this paper explains the application side of how multi-modal test execution can be used for software development project. Section 2 presents issues and necessities for multi-modal functional testing tools. Section 3 presents how multi-modal functional testing is implemented in Fitclipse and examples of how tests can be specified and executed. Section 4 concludes the paper arguing for the usefulness of having multi-modal acceptance test execution.

2. Motivation for Multi-modal Functional Test Execution

The purpose of *multi-modal test execution*, which will be abbreviated to MMTE, is to provide one-to-many mapping between the acceptance test definition and the fixture test executions: a single functional test or a feature in the requirement is executed against different layers or components of the software system. The customer may define functionality, but the actual implementation of the requirement can exist in many forms, components and locations in the implementation because a functional feature is combination of different parts of the code that works together. Rather than duplicating the acceptance test definition per appearance of the functionality or completely ignoring the multi-layer aspect of the software in the acceptance tests specifications, the next generation of acceptance testing tools should acknowledge the need for MMTE. The information must be readily available to non-developers in order for the non-developer team members and stakeholders to understand the project progress, impact of the future changes and obtain finer grained details of the implementation.

There are three benefits to MMTE.

- Finer grained progress tracking in multi-layered systems
- Respond to business requirements changes
- Derive requirements

The following sections will explain in detail how MMTE can be used to achieve the three benefits.

2.1. Progress tracking in Multi-Layered Systems

One of the best design practices for developing large software is to split the system into different layers of abstractions [8]. For example, a duplication of same code in multiple user interfaces (e.g. one web-based, one thick client and one for mobile devices) can be eliminated by abstracting out the business logic into a business layer, multiple user interface layers and other extra layers as needed (e.g. a web services layer for cross-company software integrations). In this type of multi-layered architectural software, a feature or functionality is implemented across many layers of the software architecture. For example, the business layer could be responsible for all calculations and maintaining the business rules, but the user interface layer is responsible for triggering the correct event sequences to collect the correct inputs from the user and display the output result. These two layers must be properly integrated in order for the functionality to become useful. Layered abstractions is important because it can help pinpoint failing code quickly and help fix the code without creating contradicting behaviors between different interfaces.

MMTE against a multi-layered system allows for finer grained progress tracking than “features” as it determines which layers of a feature are already working and which are failing or unimplemented. The finer grained tests can be beneficial to a wide range of audiences. For example, developers rely on automated acceptance testing for regression testing, which can help developers feel safer about changing code, especially for large or legacy applications. Having finer grained acceptance tests can help find the root cause of a bug quickly by identifying the lowest level that suddenly breaks.

In addition, understanding the design of the multi-layered software architecture could help the QAs to write better test cases for other testing procedures, especially during system testing and integration testing. A project manager can obtain quantitative figures on the progress of the development project in finer details, such as the number of failing acceptance tests and the number of unimplemented functional features. Estimating the time, budget and resources can become more accurate using the figures provided by the executable acceptance tests.

2.2. Respond to Changes

The benefit of having architectural information embedded in the acceptance tests through MMTE is quick feedback about the impact of these system-wide

architectural or business requirement changes that may occur during the development project. Customers could change their mind during the project about the requirements due to changing business requirements, budget or emergence of new technology. Or the developers could realize an additional layer of data abstraction is required or removal of a layer is more appropriate from the original specification. As the changes occur in the test definition, the exact place where code modification is required becomes apparent to everyone through failing tests. Following the spirit of agile development about "responding to change" [9], the team can improve communication to all stakeholders based on the concrete evidence provided by the MMTE. Resource allocation or feature negotiation can become easier when all stakeholders are informed with more concrete evidence to base their decisions.

2.3. Derive Requirements

MMTE can also play an important role in deriving requirements. Often customers may not be aware of exactly what they need, thus the requirements may be too vague. Having MMTE capability in the testing tool can help business analysts, QAs or developers to expand the details of the specification to achieve more concrete design requirements. Whether the details are divided in terms of different software components or different software architectural layers, having a one-to-many mapping capability can help eliminate requirements ambiguities and define more detailed requirements than just a "feature".

3. Implementation

FitClipse [10] is an executable acceptance testing tool with features such as the ability to perform acceptance test refactoring, multi-modal test execution, card-based project planning and test failure analysis using the test result history. Fitclipse attempts to address some of the critical research issues in executable acceptance testing, such as test maintenance, information abstraction and test failure analysis. Fitclipse is based on Fit [4] and FitLibrary [6] and it is built as an Eclipse plugin, which allows the developers and QAs to keep the acceptance tests in the same project location as the code.

Fit [4] is designed to facilitate automated acceptance testing using HTML tables. Fitclipse uses Wiki syntax to define the Fit tables and uses Fitnesse to convert the wiki-syntax into HTML table. A study done by Read et al. shows that over two-thirds of the computer science students who tried Fit responded positively on using Fit for future projects [12]. Similarly, the study by Melnik shows that half of the business students also responded positively on the use of Fit [13]. These studies as well as its common use also show that Fit is a good starting point for more research as it has generated a lot of strong opinions about how acceptance tests should be [13].

The customer will write the functional requirements test definition. If the functionality exists in many places in the software, the developers can expand the test definition to include MMTE. In order to provide one-to-many relationship between the acceptance test definition and the test fixtures, we decided to expand the first row of the Fit table definition to specify more than one fixture separated by a comma. There should be one Fit table, but many fixtures – each mapping the test to a specific layer or component of the SUT (System-under-Test). If there are four fixtures, then Parser will create four Fit tables. Fit will execute these four tables as if these specifications were manually specified four times by the user. After the tests are executed and the results are returned, Fitclipse will combine the results from the duplicate tables into one table and lay them side by side for easy comparison.

One Test, Three Fixtures

eg.DeparturePlane, eg.DeparturePlane_UI, eg.DeparturePlane_API			
Airline	DepartingTo	GateNumber	DepartureTime()
Air Canada	Toronto	A15	07:25
United Airlines	Denver	C20	07:27
Westjet	Los Angeles	D47	10:55

Figure 1: One test, many fixtures

Figure 1 shows an example of one-to-many mapping between test and the fixtures. In this example test, three fixtures are associated with this test.

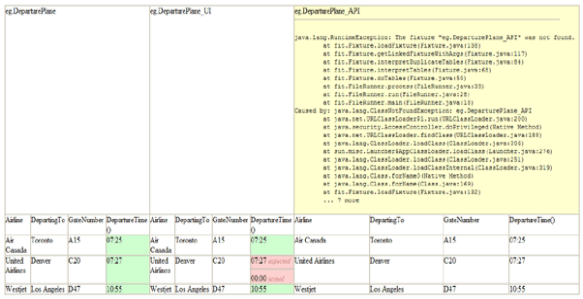

Figure 2: Execution result of the test with three fixtures

Figure 2 shows an example execution result of three test results shown side by side in order to easily convey the message about the state of the implementation. This test result shows that although the first test (business logic layer) is passing, the second test (GUI layer) is returning wrong result and

the last fixture is not implemented yet as it returns exception errors.

4. Analysis

The benefit of MMTE is the ability to convey the state of the project progress in finer details such as the existence of the same functional behavior in multiple parts of the software and having more concrete acceptance tests in order to reduce the cost due to miscommunication. For example, if the customer specifically asks for data to be accessible in two different types of database, install in multiple platforms or provide same data using two different types of APIs, MMTE is useful in conveying the requirements information more effectively.

However, it is important to note that test automation is not test-specification automation [14]. The ability to enhance the clarity of the software requirements through examples, communicate architectural design decisions to the stakeholders and be able to perform automated regression testing at multiple-layers of the code comes at a price of time and effort. However, we would argue that having this information is much better than trying to figure out what the developers did after the implementation is written or trying to fix the software with high defect rate afterward.

4. Future Work

The next step to MMTE is to perform analysis on the specific types of software development scenarios that can benefit the most from MMTE. Particularly, we believe that the test result from MMTE will have a lot of value to the development team in communicating the software architecture. We are interested to find out the adoption process of such technique, such as the obstacles in implementing the technique in practical situations and the ways in which people perceive the benefits of the technique.

There are two research problems to multi-modal acceptance testing. The first part is the MMTE and we have provided one of many solutions that are possible. The second part is the *multi-modal test definition* where different formats are better suited for representing different types of requirements information and/or for different users of the specification. Because the main author of the executable acceptance testing is the customers with little or no software engineering background, it is important to build a tool that can accommodate the way customers think about software requirements, rather than make them conform to the way software engineers and QAs think about requirements specifications and requirements testing. It would also be interesting to see the possibilities of integrating with complementary tool like Zibreve [15]. Once both testing methodologies are implemented, user studies will be performed to analyze the usage patterns and the impact as a method as a whole.

5. Conclusion

This paper presented problems with the current state of the executable acceptance test-driven development process and shows why multi-modal functional testing is useful. We described our tool that supports multi-modal test execution. Because Fit is widely used by agile teams, we used it as the basis for our own extensions and implemented the feature in Fitclipse. We added the capability to attach multiple fixtures per test definition in order to facilitate multi-modal functional test execution. MMTE helps identify which layer is the root cause of a bug and encourages a better progress tracking on a finer granularity than a “feature”. Multi-modal test execution is a method rather than just a tool implementation. It should be supported by all acceptance testing tools and everyone involved in the software development, not just the developers and QAs.

7. References

1. Beck, K.: *Extreme Programming Explained: Embrace Change, 1/e*, Addison-Wesley, Boston, MA, 1999
2. WATIR http://wtr.rubyforge.org/watir_user_guide.html
3. Selenium http://www.openqa.org/selenium/
4. FIT http://fit.c2.com/
5. FITnesse http://fitnesse.org/
6. FitLibrary http://sourceforge.net/projects/fitlibrary
7. Beck, K. *Test-Driven Development: by Example*, Addison-Wesley, 2002
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994
9. Agile manifesto. http://agilemanifesto.org/
10. Fitclipse, http://sourceforge.net/projects/fitclipse/
11. Martin, R., Melnik, G., 2008, Tests and Requirements, Requirements and Tests: A Mobius Strip, IEEE Software, 25(1), pp. 54-59
12. Read, K., Melnik, G., Maurer,F., Students experiences with executable acceptance testing, Jul 2005, Proceedings of the Agile Development Conference 2005
13. Melnik, G., Empirical Analyses of Executable Acceptance Test Driven Development, July 2007, PhD Thesis, University of Calgary
14. Martin, R., Melnik, G., 2008, Tests and Requirements, Requirements and Tests: A Mobius Strip, IEEE Software, 25(1), pp. 54-59
15. Zibreve, http://www.zibreve.com/