Fit for Business Process Testing

While low level unit testing has been quite well explored and is now established, the automated testing of complex business processes is still a challenge. One reason for this is that developers often just do not understand or speak the language of business people and vice versa. This makes the specification of test cases for real business processes more difficult. Another reason is that unit testing frameworks, which are very well suited for the automated testing of low-level classes, were not designed to test high-level processes needing extra infrastructure. In this article we will show how the open source framework Fit, which was designed for high-level acceptance testing, can be used for automated business process testing and how these tests can be added to legacy code.

Cecilia Garcia Garcia, Martin Kropp, Wolfgang Schwaiger | martin.kropp@fhnw.ch

With the increasing popularity of agile software development methods and of xUnit testing frameworks, unit testing has become very popular and can be seen as an accepted technique for automated low level component testing (units and methods). Unfortunately, this cannot (yet) be said for high-level testing on the acceptance, service, and especially, the business process level. The reasons for this lack of automated testing techniques are manifold [Fit, Mug05]:

- Business people and developers speak different languages;
- Business processes are complex and have more dependencies on external resources than low level components;
- The number of available automated testing tools is limited.

Different languages: Business people and developers are experts in their own application domains, each with their own domain language and terms. However, these different domains often do not understand one another either because of the different terms, or even worse, because of having the same expressions but with different meanings, which can cause fundamental misunderstandings.

Complexity: Business processes usually consist of a sequence of interactions between the user and the system under development, often with alternative control flows. Furthermore, business processes often require additional infrastructure, such as servers, databases, and communication channels, which at least can cause extra effort when implementing tests.

Automated Testing: Various tools for automated acceptance testing, the so-called Capture-Replay tools, are available (e. g. [jRapl]). However, they are typically bound to GUI controls and sometimes even to their absolute position in the window, which makes them prone to changes. Thus, a good support tool for the testing of high-level services and business processes is still missing [Mens07].

In this paper we will describe how the Framework for Integrated Testing (Fit), can be used to specify and implement automated business process tests for existing code and to help solve at least parts of the above mentioned problems [Fit].

Fit is an open source acceptance testing framework, which was developed by Ward Cunningham et al. Its goal is to bridge the gap between users and developers and to support automated testing on the acceptance level, thus addressing the above mentioned problems.

Case Study

We will demonstrate the testing of business processes by using the Java Pet Store web application [SunPet]. This application was chosen as a demonstrator, because (1) it contains the typical business processes of shopping applications, (2) it has no implemented test cases, in fact, it was not even designed for testing, (3) it shows the typical characteristics of a complex software product, such as the usage of different technologies for development (AJAX, J2EE, JSF), a complex structure, a database connection, legacy code, and (4) it is easy to understand from a user's point of view.

Figure 1: The «Pet Store» application

We will use the common and well-known «search-select-buy» shopping process of an online store.

The Shopping Business Process

When users enter the Java Pet Store application, (see Figure 1), they can choose to search for a pet. Then the search criteria is entered, which can include the kind of pet, parts of a description, and additional criteria (tags). Once the search has been completed, the results are displayed on a catalogue screen.

The user then continues the selection of the desired pet. In the catalogue, the user chooses the buy option using the external PayPal [PayPal] payment service system. The individual steps in the process can be seen below:

1.	Select search menu option
2.	Enter search criteria
3.	Check/Uncheck "Also Search Tags"
4.	Execute search
5.	Present search results
6.	Select desired pet
7.	Buy pet with the PayPal option

Table 1. The pet shopping process

Writing Test Cases

With Fit, once the business process has been specified, the user continues to write the test cases. As mentioned before, Fit tries to bridge the gap between users and developers by allowing the user to write the tests in his or her own domain language and from his or her own perspective.

To do this, Fit uses the concepts of *tables* and *fixtures* to specify the tests for *the system under test* (SUT) (see Figure 2). Fit reads the specified test cases from one or several HTML tables, interprets this data according to the underlying test type (fixture type), and then executes the tests on the SUT [Mug05].

This concept allows an automated, early and immediate feedback loop between users and developers, thus preventing the problem of late acceptance and business process testing.

Tables

Tables represent the user's view of the tests. Assuming that users and business people know how to deal with tabular representations in general, a tabular form for test case specification makes it easy for them to write tests.

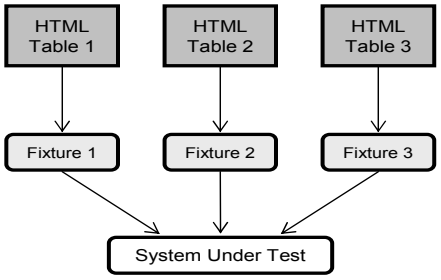

Figure 2: Fit concepts

Fit uses HTML tables to specify the tests and to report the test results. To indicate the success or failure of the tests, Fit uses a traffic light metaphor, which is well-known from other unit-testing frameworks [JUnit]. Moreover, Fit not only allows specifying expected data but also to specify commands and expected behavior and thus simulates user interaction and complete business processes. An example for the latter is given in Table 2, where the search-process of our case study with its input data and its expected behavior is specified.

To make it even more convenient for users and business people, tables can be written using other tools like Microsoft© Office (the generated files, however, have to be stored in *.html format). Furthermore, the wiki-based platform FitNesse [FitNesse] even allows the specification and execution of acceptance tests in distributed project teams over the web.

fit.ActionFixture		
start	Test search submit	
enter	search string	cat
enter	search tags	true
press	submit	
check	number of results	2

Table 2. Test cat search process

In general, the first row of a table always specifies the type of test to be executed, which is called a *fixture type* in Fit terminology (see table 2). The content of the subsequent rows depends on the fixture type and ranges from pure input data and expected data to commands and instructions as shown above. In the following, we will give an overview of the basic fixture types (a complete description can be found under [Mug05]).

ColumnFixture tables are used to test pure calculations and are the simplest type of table. Table 3 shows an example of a discount calculation test, which checks the correct discount percentage and the correct discount price calculation. It uses two input parameters (amount and customer category) and two expected values for comparison (percentage, discount price).

The first row specifies the name of the fixture, the second row the names of the input fields (amount, customer category) and the calculated values (percentage(), discount price()). Each following row of the table represents an independent test case.

Calculate discount			
amount	customer category	percentage()	discount price()
0.00	Gold	30%	0.00
120.00	Gold	30%	84.00
120.00	Silver	20%	96.00
999.00	Regular	0%	999.00

Table 3. ColumnFixture sample

Test search for available cats			
name	description	tags	price
Friendly Cat	This black and white colored cat is super friendly. Anyone passing by your front yard will find him puring at their feet and trying to make a new friend. His name is Anthony, but I call him Ant as a nickname since he loves to eat ants and other insects.	awesome interesting cool	307.10
Fluffy Cat	A great pet for a hair stylist! Have fun combing Bailey's silver mane. Maybe trim his whiskers? He is very patient and loves to be pampered.	awesome interesting cool	527.00

Table 4. Test database query result with RowFixture

RowFixture tables are a special type of ColumnFixtures. While the rows of the ColumnFixture table are executed independently of each other, rows of a RowFixture table build an entity and can represent a group, a list, a sequence, or a collection of things. As an example, Table 4 shows our «search cat» case study test.

ActionFixture tables are typically used to simulate actions, performed by the user on a screen, for example. Moreover, ActionFixture types are very well suited for testing even large, sequential processes. An example of an ActionFixture can be seen in Table 2. The left column contains the actions to be executed. Fit offers the following predefined action commands: *start*, *enter*, *press*, and *check*. The second column describes the action to be executed in the user's language, and the third column contains optional input parameters for the action.

Testing a Complete Business Process

We have seen how individual test cases can be specified very easily with tables, now we are going to specify the test for the complete shopping process as shown in Table 1. Fit addresses these requirements by combining several tables to a single HTML page.

We want to test the complete shopping process with the following tests and we will use the specified fixture types. The complete test process can now be easily written by putting all the different test tables into a single webpage as shown in Figure 3.

	Test	Description	Fixture Type
1.	Test Search Process	Test if the specified test criteria deliver the correct amount of data	ActionFixture
2.	Test Search Result	Test the returned result set for correctness	RowFixture
3.	Test Selected Product	Test if the correct product is read based on the product's ID	ActionFixture
4.	Test Payment	Check if the payment process works correctly	RowFixture
5.	Test Catalogue after selling a pet	Test if the sold cat is removed from the catalogue	RowFixture

Table 5. Test cases for the shopping process

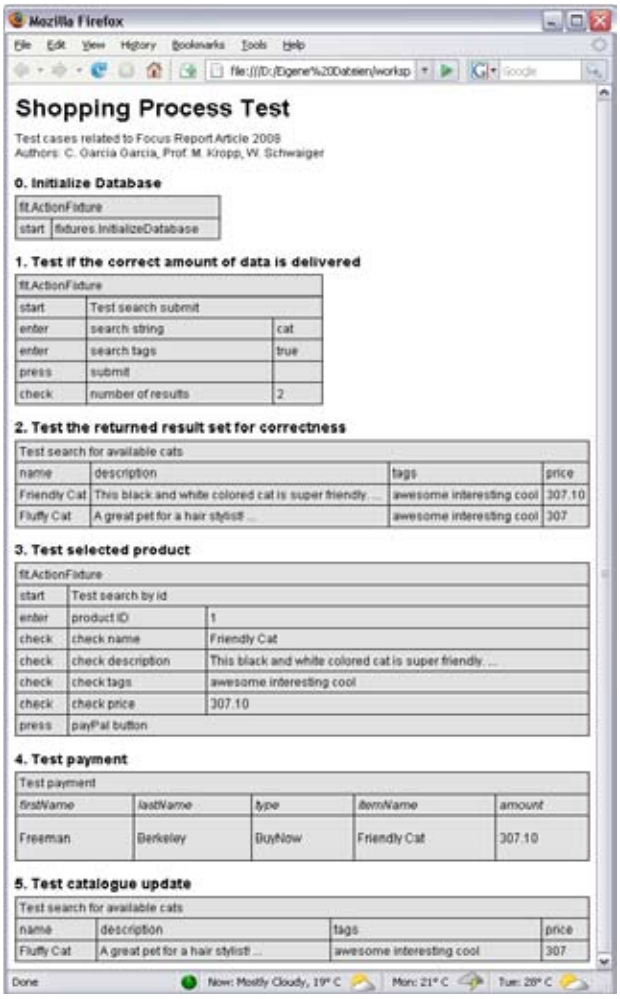

Figure 3: Complete shopping process test in HTML

In this special test case we expect that the user has found the pet of his or her choice and wants to buy it. A pet has been selected from the search result and then the user is redirected to the store's catalogue where the selected pet is pictured. On the catalogue screen the «PayPal» icon is selected to pay for the new mate. After payment has been made the purchased pet must be removed from the catalogue, which is then tested as seen in Figure 3.

If the Fit test cases are simple, then the table will be stand-alone and self-contained. This is not the case for multi-table test cases. In a test process, subsequent tables usually depend on the preceding ones. This requires that the test case planner and the test case implementer work closely together, to clearly communicate the dependencies which cannot be seen from the tables themselves.

Writing the Test Code / Binding Tables to Code

After the test cases have been specified in the user's own domain language, the developer can implement the fixtures. This glue code differs according to the implemented fixture type. Listing 1 shows the fixture code for second table from Figure 3.

The relevant statements have been highlighted. The first row in the table specifies that an action fixture will be used, i.e. that the first column of the

```
public class TestSearchSubmit extends Fixture {

    private ArrayList<Item> hitlist;

    public int numberOfResults() { return hitlist.size(); }

    public void searchString(String searchString)  {
        fixtures.InitializeDatabase.dataBaseMock.setSearchString(searchString);
    }
    public void submit() {
        hitlist = fixtures.InitializeDatabase.dataBaseMock.searchAction();
    }
    public void searchTags(boolean searchTags) {
        fixtures.InitializeDatabase.dataBaseMock.setSearchTags(searchTags);
    }
}
```

Listing 1: From test table to code

following rows contains actions to be executed. The second row specifies the name of the concrete test class (which is derived from the class Fixture). All subsequent rows contain the concrete actions to be executed and the necessary data. While the first column of these rows specifies the action type, the second column corresponds to the appropriate method in the test class, and the third (and, optionally, all the following columns) specify its input data. Fit allows a very readable form for the names in the table and automatically converts them into the appropriate classes, methods and variables in the fixture by removing punctuation and blanks and by using camel casing [FitGrace].

The third table is a RowFixture and its important implementation details are shown in Listing 2. A RowFixture contains two important methods; *getTargetClass()* and *query()*. The getTargetClass() method allows Fit to retrieve the concrete instance of the RowFixture class, which is then used to call the query()-Method. The query()-method assembles the elements of the actual list from the result list returned by the database and forwards it to Fit for comparison. In the PetObject class in Listing 2, it can be seen that the number of defined attributes does not exactly match the column count of the third table in Figure 3. Fit does not directly compare the fields of the returned query to a table row; it really adheres to the specified headings and attributes. The implementation of the remaining tables is equivalent to the just described code listings.

Conclusion

Writing automated business process tests poses additional challenges for the test specification and implementation.

In this article, we have shown how the Fit testing framework can be used to test business processes. Its approach of using tables to specify tests, independently of the actual GUI, makes it easy for business people and users to write test cases in their own domain language and thus it can indeed help to bridge the gap between developers and users. This makes Fit a perfect complement to xUnit testing frameworks used to test low level classes and methods.

In our case study we concentrated on the Fit core library. This library makes it very easy to write and run tests and to demonstrate the basic principles of business process testing. For more complex processes, however, the data setup and the test tables become cumbersome and hard to read [Mug05]. Moreover, to share a common context between multiple tables, the Fit core library uses the not terribly OO-like approach of public static members.

To avoid these problems, additional fixture types have been developed for the Fit Library [FitNesse, FitLib], which offers, for example, the fixture types SetUpFixture and DoFixture to handle setups, workflows and to share a common context between tables.

The open architecture of Fit enables the customization and the extension of its functionality to meet individual needs. The existing command set of ActionFixtures can be easily extended through the subclassing and the addition of a new method for each new command. To define a custom command set in a separate class, a new action class can be directly derived from the class Fixture and new commands implemented as new methods.

Moreover, developers can define their own fixture types by just deriving from any corresponding base fixture class. This allows the development of domain specific fixture types such as test, database, GUI and service [FitFix]. To enable fully automated regressions tests, Fit can be run from a command line.

Although there is room for improvements in Fit, e. g. usability and ease of use, but also the refactoring of existing test cases [Otting08], plug-in support and extensions [FitPro], Fit does well when acceptance and business process test-ing are considered, especially when it is used from the very beginning of a software project.

```
public class TestSearchForAvailableCats extends RowFixture {

    public Class<?> getTargetClass() { return PetObject.class;  }

    public Object[] query() {
        ArrayList<Item> resultList = fixtures.
            InitializeDatabase.dataBaseMock.searchAction();

        PetObject array[] = new PetObject[resultList.size()];

        for (int i=0; i < resultList.size(); i++) {
            array[i] = new PetObject(
                resultList.get(i).getProductID(),
                resultList.get(i).getName(),
                resultList.get(i).getDescription(),
                resultList.get(i).tagsAsString(),
                resultList.get(i).getPrice().toString(),
                resultList.get(i).getImageURL(),
                resultList.get(i).getImageThumbURL());
        }
        return array;
    }
}

// the Pet business object
public class PetObject {
    public String productID;
    public String price;
    public String name;
    public String description;
    public String tags;
    public String imageURL;
    public String imageURLThumb;

    public PetObject(String productID, String name, String description,
        String tags, String price, String imageURL, String imageURLThumb) {
        this.productID = productID;
        this.price = price;
        this.name = name;
        this.description = description;
        this.tags = tags;
        this.imageURL = imageURL;
        this.imageURLThumb = imageURLThumb;
    }
}
```

Listing 2: From test table to code 2

References

[Ambler08] S. W. Ambler, April 2008, Introduction to Test Driven Design (TDD). http://www.agiledata.org/essays/tdd.html.

[Beck08] K. Beck, April 2008, XProgramming.com - an Agile Software Development Resource. http://www.xprogramming.com.

[Fit] W. Cunningham, et al., April 2008, Fit: Framework for Integrated Test. http://fit.c2.com.

[FitClips] IBM, Vishnu Vettrivel, April 2008, FIT and Eclipse: Testing with the Extended FIT Eclipse plug-in. http://www.ibm.com/developerworks/aix/library/au-fiteclipse/index.html.

[FitFix] FitNess Useful Fixtures, April 2008, http://fitnesse.org/UsefulFixtures

[FitGrace] Improving Readability using Graceful Names, April 2008, http://fitnesse.org/FitNesse.GracefulName.

[FitLib] FitLibrary, April 2008, http://fitnesse.org/FitLibrary.

[FitNesse] FitNesse, April 2008, www.fitnesse.org.

[FitPro] SourceForge.net, April 2008, FITpro - Acceptance Test Solutions. http://sourceforge.net/projects/fitpro/.

[jRap] John Steven et al. jRapture: A Capture/Replay Tool for Observation-Based Testing. Intl. Symp. on Software Testing and Analysis, Portland, Oregon, August 2000, pp. 158-167.

[JUnit] JUnit.org Resource for Test Driven Development: JUnit.org, April 2008, http://www.junit.org.

[Mens07] Tom Mens, Serge Demeyer: Software Evolution (pp173), Springer 2007.

[Mock08] Mock Object Authors, April 2008, Mock Objects. http://www.mockobjects.com.

[Mug05] R. Mugridge, W. Cunningham: FIT for Developing Software. Framework for Integrated Tests. Prentice Hall International, 2005

[Otting08] Tim Ottinger's Blog on FitNesse, April 2008, http://tottinge.blogsome.com/2008/02/01/fitnesse-frustration/.

[PayPal] PayPal: Online Payment, Merchant Account - PayPal. April 2008, https://www.paypal.com.

[SunPet] Sun Microsystems, Inc., April 2008, Java™ Pet Store Demo. https://blueprints.dev.java.net/petstore.