PROVIDING FACILITIES FOR THE USE OF TDD IN PRACTICE

Vinícius Pereira and Antonio Francisco do Prado
Department of Computing, Federal University of Sao Carlos, Sao Carlos, Brazil

Keywords: Functional Testing, Test Case Generation, Code Transformation, Test Driven Development, User Stories, Test, User Acceptance Testing, Scrum.

Abstract: In this paper we describe an approach that provides a way to facilitate the use of TDD in the practice of Web application development, independent of the development process used. As an example, it is presented a way of integrating the functional testing (on level of acceptance) in the Scrum process. The functional tests are constructed to test the User Stories, which represent the software requirements specified in the Product Backlog according to the Scrum. The approach is divided into three stages: Tests Specification, Functional Tests Construction and Tests Driven Implementation. An example of social application illustrates the use of the proposed approach.

1 INTRODUCTION

In the context of software engineering, software testing is considered a priority activity (Bertolino, 2007). This is due to the fact that a defect found when the software is in production may have a much higher cost to fix than if the same were found in the early stages (Perry, 2006).

In this paper we present an approach that aims to provide facilities for the practical use of Test Driven Development (Beck, 2003) in software development. To exemplify the use of this approach is shown how to integrate functional testing with the Scrum process, aiming to combine the advantages of agility and knowledge of this agile method to the of Tests which aim to reduce the mistakes of inconsistent implementations with the requirements specified for the software.

The proposed approach, which emerged during the RAMBUS process improvements (Pereira and do Prado, 2011) (a variation of Scrum), of the same authors, aims to improve software quality without losing the agility of the Scrum. Based on the User Stories (Cohn, 2004), the integration of functional testing helps the development team on the task of implementing the software in a manner more consistent with their requirements. For the tests to simulate the functionality of the software, a test specification language was developed and a converter was built to generate, based on these specifications, the test code in the target language of the software implementation.

In the following section we present the proposed approach. And the section 3 presents the conclusions and future work.

2 THE PROPOSED APPROACH

The explanation of the proposed approach shows how to integrate its three stages with the Scrum process to create functional tests. The three stages are: Test Specification, Functional Test Construction and Test Driven Implemantation. These stages are present in each Scrum Sprint, as shown in Figure 1, and are performed for each of the User Story.

The approach is based on the TDD concepts of "to create a test, check its failure, write the code for the test to be approved" and the concept to develop driven by the expected behavior of software. To assist the understanding and the inter-relationship of these three stages, Figure 2 shows in a SADT diagram (Structured Analysis and Design Technique) (Ross, 1977) the inputs and the outputs of each stage, as well as their respective mechanisms and controls.

In the following, the stages of the approach are presented in more detail. To facilitate understanding, it is used as an example the development of an application domain of social network. This application integrates Facebook[1] and Google+[2] features and

[1]https://www.facebook.com/
[2]https://plus.google.com/

Pereira V. and Francisco do Prado A..
PROVIDING FACILITIES FOR THE USE OF TDD IN PRACTICE.
DOI: 10.5220/0003906302420245
In *Proceedings of the 8th International Conference on Web Information Systems and Technologies* (WEBIST-2012), pages 242-245
ISBN: 978-989-8565-08-2

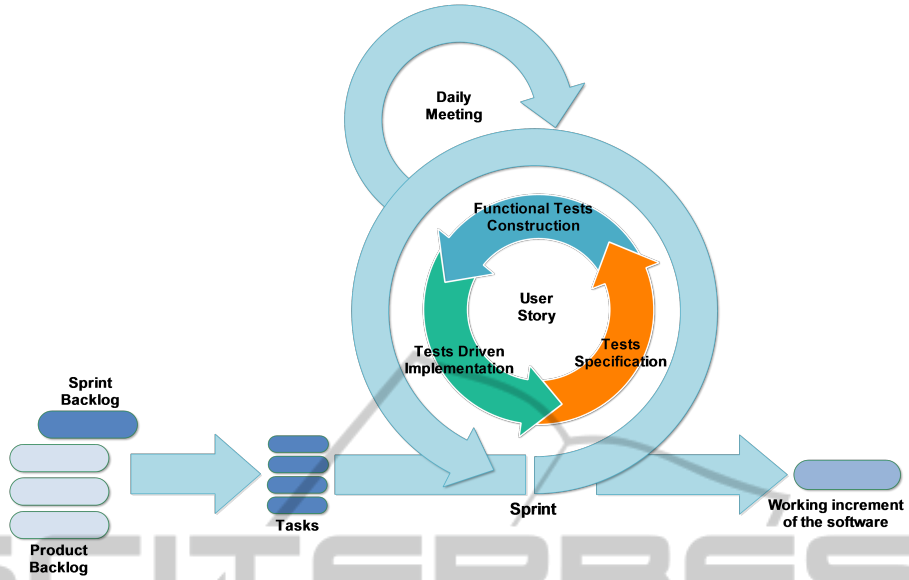

Figure 1: Scrum and Functional Testing.

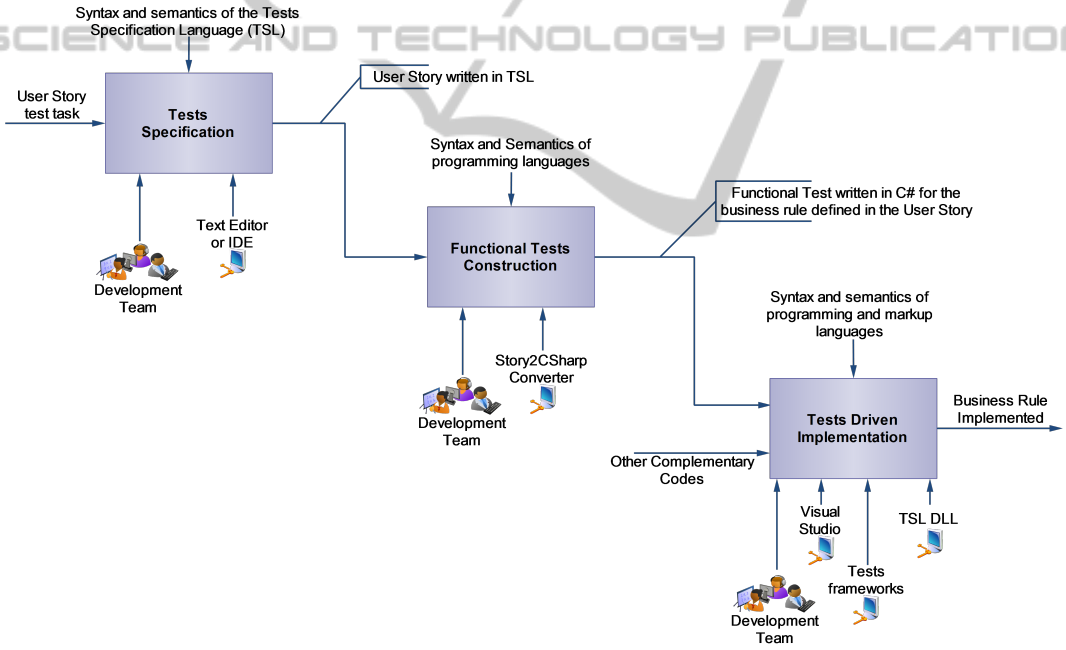

Figure 2: SADT of the proposed approach.

aims to be a social networking site with a focus on build and reflect social networks and social relations between people. The functional tests created in this example are in the acceptance testing level.

Tests Specification: at this stage, based on the Story Cards and Mockups, the development team should write the User Story using the Test Specification Language (TSL).

Created by the authors, this language has some minor modifications made to the structure of the User Story provided by Mike Cohn (Cohn, 2004), towards implementing in C# (the chosen target language). Instead of double quotes (“ ”), the symbol “$” was used to reference identifiers (fields, variables, and buttons) and braces ({ }) were used to indicate strings. If necessary, the TSL grammar can be easily modified, making it more appropriate to specify the User Stories. The Figure 3 shows an example of an User Story for a social network application defined using TSL.

```
Feature: New user registration
In order to use MySocial
As a new user
I want to register me

 Scenario: Sign up
  Given I go to the new user registration page
  When I fill in $username with {testuser}
  And I fill in $email with {test@test.test}
  And I fill in $user_password with {secretpwd}
  And I fill in $password_confirmation with {secretpwd}
  And I press $create_my_account
  Then I should be on the getting started page
  And I should see {Welcome}
  And I should see {Fill out your profile}
  And I should see {Connect with cool people}
  And I should see $Finished
```

Figure 3: User Story written in TSL.

Functional Tests Construction: in this second stage, the development team uses the User Story written in TSL created in the previous stage to create a business rule instrumented with functional test. This stage is aided by Story2CSharp Converter, which converts the TSL for the C# language. This converter was developed by the authors to support the approach together with the TSL.

Operationally, the development team copies the contents of the User Story described in TSL (or even write the contents) to the left screen of the Story2CSharp Converter. The converter generates the C# code in execution time and displays it on the right screen. The Figure 4 shows the converter with a iteration button created when needed. In this moment, with the User Story being written in the converter screen, the next line should begin with the keyword "Given". The Figure 5 shows the conversion result of the User Story in Figure 3 to the functional test in C#.

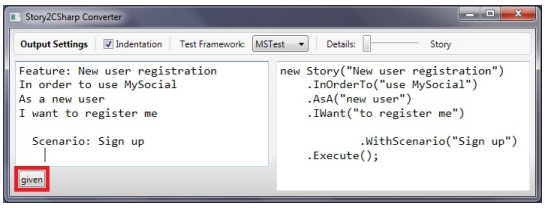

Figure 4: The Story2CSharp Converter in action.

As shown in Figure 5, it was generated a test class with the name indicating the use of the library Story2CSharp and each story becomes a testable method. It is important to note that although converted to C#, the Story is perfectly legible in order to maintain a level of abstraction close to that offered by TSL.

Next is created a variable Story which will contain the information of the User Story. The contents of each scenario is divided into methods that are declared externally in relation to the testable method.

```
using System;
using Story2CSharp;
using Microsoft.VisualStudio.TestTools.UnitTesting;

[TestClass]
public class Story2CSharpTestClass {
  [TestMethod]
  public void NewUserRegistration() {
    new Story("New user registration")
      .InOrderTo("use MySocial")
      .AsA("new user")
      .IWant("to register me")
          .WithScenario("Sign up")
            .Given(IGoToTheNewUserRegistrationPage)
            .When(IFillIn_With_,"username","testuser")
              .And(IFillIn_With_,"email","test@test.test")
              .And(IFillIn_With_,"user_password","secret")
              .And(IFillIn_With_,"password_confirmation","secret")
              .And(IPress_,"Create_my_account")
            .Then(IShouldBeOnTheGettingStartedPage)
              .And(IShouldSee_,"Welcome")
              .And(IShouldSee_,"Fill out your profile")
              .And(IShouldSee_,"Connect with cool people")
              .And(IShouldSee_,"Finished")
      .Execute();
  }

  private void IGoToTheNewUserRegistrationPage()
  { throw new NotImplementedException(); }

  private void IFillIn_With_(string arg1, string arg2)
  { throw new NotImplementedException(); }

  private void IPress_(string arg1)
  { throw new NotImplementedException(); }

  private void IShouldBeOnTheGettingStartedPage()
  { throw new NotImplementedException(); }

  private void IShouldSee_(string arg1)
  { throw new NotImplementedException(); }
}
```

Figure 5: User Story, of the Figure 3, as functional test in C#.

The name of each method is defined to indicate with an underscore (_) where their attributes fit the name for easier its reading. These methods are designed with the goal of throwing an exception to indicate that they have not yet been implemented. At the end of the testable method, an internal function is called to execute the Story.

Thus, the development team builds the functional test integrated with the User Story. Note that the business logic is not automatically generated. Only the structure of the test is generated, leaving the writing of the logic to the next stage of the approach.

Tests Driven Implementation: in this last stage, the functional test generated in the previous stage should guide the coding of business logic, following the principles of BDD (North, 2006). Other codes may have been created automatically or not, to support this stage of the approach, such as the creation of CRUD (Kilov, 1998) and the database.

In the Visual Studio environment, the approach is supported by VS Unit Test and NUnit testing frameworks as well the library Story2CSharp. The development team includes the code generated by the converter in the project created in Visual Studio.

Once the functional test was constructed, the development team can execute it and analyze its results. Because the code has just been generated, the most likely outcome is that the functional test is "inconclusive", since the methods of each scenario are yet structures that were not implemented. The Figure 6 shows this result.

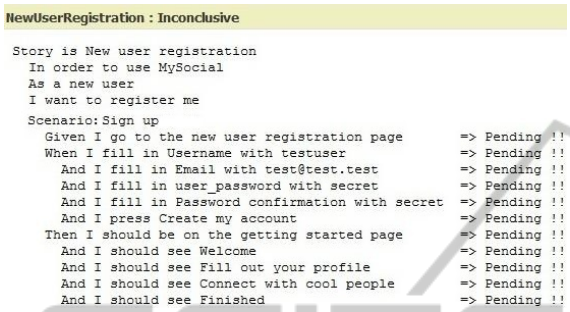

Figure 6: Result showing that the functional test was inconclusive, due to pending methods.

Following, the development team writes the code to test the business logic of the User Story so that its steps accuse failures. To correct the failures, the development team implements the business logic for the steps that be approved in the tests.

The use of this approach does not preclude the use of unit testing. The use of these tests help to complement the evaluation of the User Story. A way to integrate the unit testing with functional testing can be found in the Chelimsky's book (Chelimsky et al., 2010).

The final result is the business logic implemented, functionally tested, according to the requirements specified in the User Story. Similarly the approach is repeated for the others User Stories in each Scrum Sprint.

3 CONCLUSIONS

The proposed approach helps in software creation by guiding the development based on the behavior expected by the user. This is possible through the implementation of the User Stories written in a Tests Specification Language (TSL), which has a high level of abstraction. These stories are the basis of the approach, describing each business rule as a Story Card and ending as Functional Tests. These tests are in the acceptance testing level and are executed to guide the implementation. This approach can be used by any development process, regardless of whether or not Agile. It is only necessary that the development team makes use of concepts and techniques of Test Driven Development.

The approach was tested in social networking applications, as the presented on this paper. Two tools were developed to support the approach: the Tests Specification Language (TSL), written as a DLL to be used with the Visual Studio and the Story2CSharp Converter, which uses the DLL and assists in achieving the functional test from the User Story written in TSL.

Future work includes case studies to test the proposed approach. Also includes the improvement of tools for importing directly from C# code for Visual Studio and the realization of new case studies in other fields of application.

ACKNOWLEDGEMENTS

The authors would like to thanks to the people from their laboratory for their support and cooperation. Thanks also to The National Council for Scientific and Technological Development (CNPq) for financial support which enabled this study.

REFERENCES

Beck, K. (2003). *Test-Driven Development by Example*. Addison-Wesley, first edition.

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *Future of Software Engineering*, pages 85 – 103, Washington, DC, USA.

Chelimsky, D., Astels, D., Dennis, Z., Hellesoy, A., Helmkamp, B., and North, D. (2010). *The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf, beta edition.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, first edition.

Kilov, H. (1998). *Business Specifications: The Key to Successful Software Engineering*. Prentice Hall, first edition.

North, D. (2006). *Introducing Behavior Driven Development*. Better Software, first edition.

Pereira, V. and do Prado, A. F. (2011). Rambus: An agile process for developing web applications. *Journal of Intelligent Computing*, volume 2:42–53.

Perry, W. (2006). *Effective methods for software testing*. John Wiley and Sons, third edition.

Ross, D. T. (1977). *Structured Analysis: A Language for Communicating Ideas*, pages 16–34. IEEE Transactions on Software Engineering 3(1).