Towards Model-based Acceptance Testing for Scrum

Renate Löffler, Baris Güldali, Silke Geisen
Software Quality Lab (s-lab), University of Paderborn
Warburger Str. 100, Paderborn, Germany
[rloeffler|bguldali|sgeisen]@s-lab.upb.de

Abstract

In agile processes like Scrum, strong customer involvement requires techniques to support requirements analysis and acceptance testing. Additionally, test automation is crucial, as incremental development and continuous integration need high efforts for testing. To cope with these challenges, we propose a model-based technique for documenting customer's requirements using test models. These can be used by the developers as requirements specification and by the testers for acceptance testing. We use light-weight and easy-to-learn modeling languages. Based on the test models, we generate test scripts for FitNesse and Selenium, which are well-known test tools in the agile community.

1 Motivation

Over the last years, agile methods have become more popular. They have been developed with the aim to be able to deliver software faster and to ensure that the software meets the customer's changing needs [1]. They all value the agile manifesto and share some common principles: Improved customer satisfaction, adopting to changing requirements, frequently delivering working software, and close collaboration of business people and developers [1].

At the moment, Scrum [7] is the most used method; about 50 percent of business companies are using the method itself or a hybrid of Scrum and XP [11]. The main focus in Scrum is on implementation rather than on a detailed analysis or proper documentation. The communication of requirements and the definition of new desired features, which are added to the existing product backlog, occur during the iterative cycles which are called *sprints* [7]. This can lead to incomplete and inconsistent specifications, which cause problems if software has to be contractually accepted by the customer. Thus, it is a challenge to establish better requirements specification techniques that overcome the addressed difficulties.

Another challenge in Scrum is the continuous integration of new functionalities during the sprints. Besides unit, integration, and regression testing, acceptance tests have to be conducted in every sprint to ensure that the product meets the required functionalities. Depending on the number of sprints, testing efforts can explode. Thus, there is an urgent need for automating testing activities, e.g. test case generation, test execution, and evaluation of test results.

In this paper, we describe how to integrate a model-based technique into the Scrum process in order to cope with the challenges addressed above without delimiting agility. First, we want to improve the communication and the documentation of customer requirements by using simple UML [5] models. These models are created together with the customer and they describe the system functions on an abstract level. The modeling languages we use are so intelligible that we believe that the customer will be able to learn and edit them easily. Afterwards, these initial models are refined during the sprint planning with technical details and test data. Then, these models are used by the developers as a specification of required functions. Secondly, we want to automate the testing activities by generating test scripts from the models. The refined models contain enough information for generating test scripts for FitNesse [4] or Selenium [8], which are well-known and widely used acceptance test tools in the agile context.

With our testing approach we follow the following ideas of Utting and Legeard [10]: (1) model-based testing must be embedded into agile processes and existing tool chains; (2) there is a need for dedicated domain specific modeling languages for making behavioral modeling easier and user-friendly; (3) model-based testing must be linked to the requirements analysis.

2 Scrum

Scrum is an empirical agile project management framework [7], which implies roles, artifacts, and activities which are mainly meetings. Figure 1 shows the Scrum process based on [2]. The elements in boxes (IOD, etc.) will be introduced in section 3.

The main artifact in Scrum is the *product backlog*, which is a prioritized feature list. The *product owner* creates, controls, and manages the backlog.

After creating the product backlog, the main Scrum flow starts. The product owner and the *Scrum team*

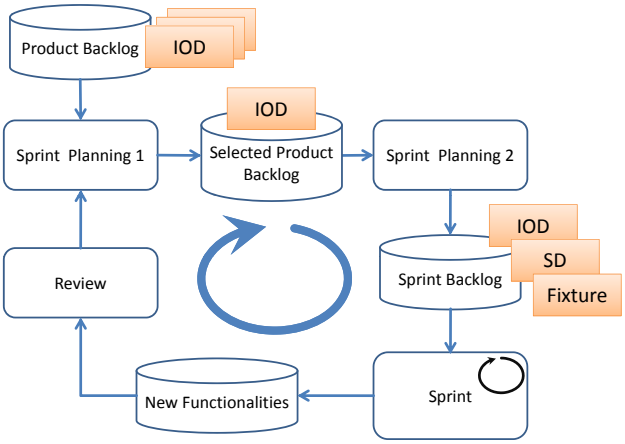

Figure 1: Model artifacts embedded into Scrum

meet for the planning of the first iteration, called *sprint*. The first features are chosen from the product backlog, and the team splits the features into smaller tasks, which results in the *sprint backlog*. At the end of each sprint, new functionalities are delivered and presented in the *review meeting* to the product owner.

During the sprint, many testing activities have to be conducted: (1) new functionalities have to be tested for correctness (unit testing); (2) the correctness of old functionalities has to be re-tested if changes were done (regression testing); (3) the integrated system of new and old functionalities has to be tested (integration testing). If these activities are conducted on the end-user interface (e.g. graphical user interface) and if they test the software against the user requirements, we speak of *acceptance testing*. Before the sprint has been finished, testers have to assure that the new functionalities integrated into the software work correctly.

In order to efficiently conduct the testing activities listed above, test automation is needed. Next section will explain how we want to automate test case generation and test execution by using abstract models.

3 Model-based Acceptance Testing

We extend the Scrum process with a model-based testing approach, where models are used for capturing customer requirements and for acceptance testing. Thereby, we want to reach a two-step improvement for Scrum: First, we want to systemize *test design* by capturing customer requirements in forms of models, which subsequently are used for test case generation. Secondly, we want to automate *test execution* by using a suitable test tool, e.g. FitNesse or Selenium.

For the first point, we add new model artifacts to the Scrum process as shown in Figure 1. As modeling notations, we use *Interaction Overview Diagrams* (IOD) and *Sequence Diagrams* (SD) defined in UML 2 [5] where the required functionalities are specified as *user stories*. User stories are suggested as a requirements engineering technique for Scrum [2]. These notations are light-weight and easy-to-learn for the customer or the product owner to create the user stories. In addition to IOD and SD, we define a *Fixture* [4], which is a domain-specific language for specifying the interface between the user and the test automation tool.

Secondly, we refine the sprint activities by using the new model artifacts as shown in Figure 2. Here, we have two roles in the Scrum team: developer and tester. The developer uses the IODs and SDs as a specification of the required functionalities. He extends the SDs with an *Interface Specification* (IS) and implements the required functionalities. The Tester extends the SDs with *Test Data* (TD). All model artifacts are used for automatic test case generation resulting in test scripts that can be automatically executed on the *system under test* (SUT).

In the following, we want to give more details on the two steps of our approach and illustrate them by using a small online store example.

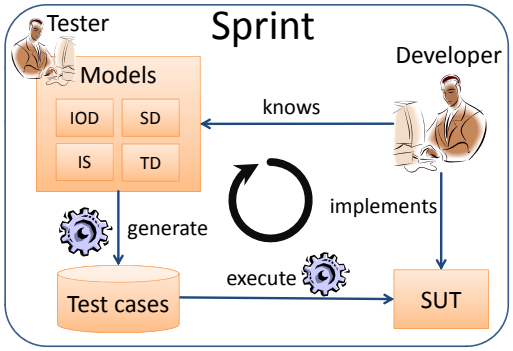

Figure 2: MBT activities conducted in a Sprint

3.1 Creating Models

First, the product owner creates user stories, which are specified with IODs (Figure 1). In Figure 3.a, the IOD for a user story describing how to buy a book in an online store is shown. Every node is given a name that describes the interaction in the user story. The product owner thus describes the backlog items on a high level, while still specifying the order of activities and their dependencies.

In the first Sprint Planning, the product owner and the team choose which backlog items they want to realize in the first sprint. Here, the IODs help in prioritizing nodes or branches and selecting them for the sprints (e.g. *OpenAmazonBookLink, BuyItem, Login, Confirmation*).

The second Sprint Planning is used to refine the selected Product Backlog Items. In our approach, this

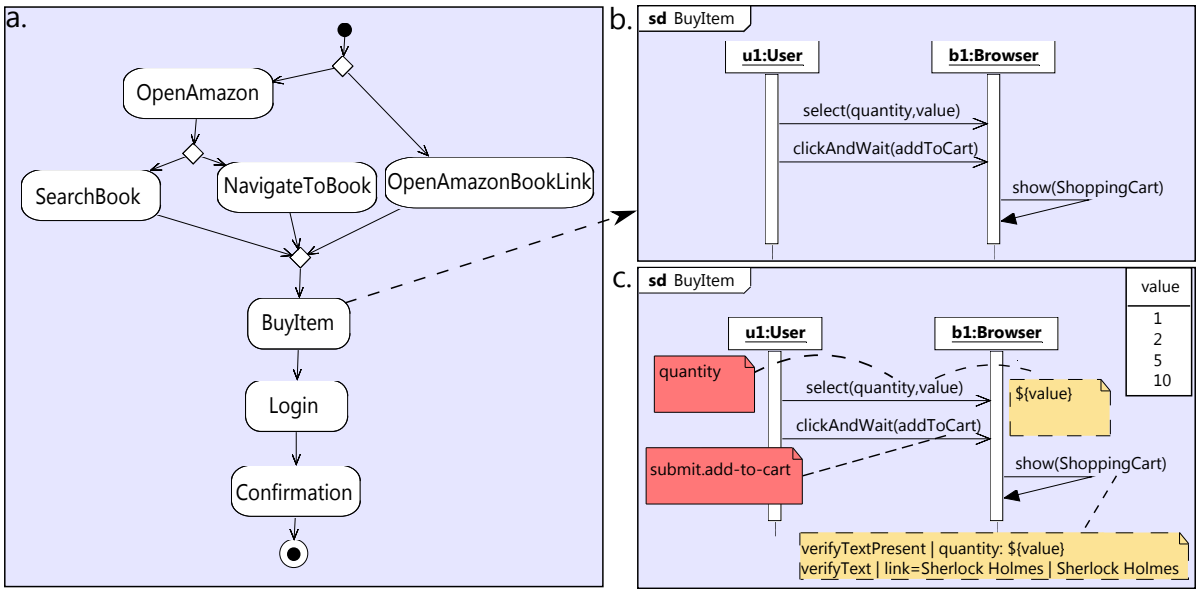

Figure 3: a. IOD describing a user story, b. Every node is linked to an SD, c. SD with annotations

means the team refines the nodes in the IOD by adding SDs to them. Figure 3.b describes the interaction in the node *BuyItem*, that is the user selects the quantity and clicks on the shopping cart button. The browser then shows the shopping cart.

Before starting to design the SDs, the team should first decide on a Fixture which defines a unified domain-specific language to specify the interactions. Since our example is about an online store, we use a Fixture for web-based interaction defined by Selenium [8] that defines keywords (e.g. *select*, *clickAndWait*) to express actions on a web page. These keywords should be then consistently used as message names for the SD.

The result of the second Sprint Planning is the Sprint Backlog, which are the selected IODs, the designed SDs, and the chosen Fixture.

3.2 Implementation and Testing

After having documented functional requirements as models, the Sprint can start, meaning the Scrum team implements the Sprint Backlog. Developers and testers can work in parallel. While the developers meet technical decisions like choosing libraries or specifying the interface (IS), testers can focus on the selection of test data (TD). The IODs and SDs are used as *test models* by the testers for generating test scripts.

The IS comprises all information that is specific to the implementation, e.g. a specific field name or button name. In Figure 3.c, the solid boxes belong to IS, which specify the name of the select field as *quantity* and the name of the submit button as *submit.add-to-cart*.

The TD consists of the test input and some verification steps as shown as dashed boxes in Figure 3.c. For selecting test inputs, well-known techniques can be used (e.g. equivalence classes, boundary analysis). In the example, the TD for the quantity contains the values 1, 2, 3, 10. The expected results are specified using the Fixture language. The verification step in Figure 3.c checks whether the quantity given by the user appears correctly in the shopping cart.

For automating test execution, we are developing a prototype test case generator as a plugin for Fujaba4Eclipse [9]. Test scripts are generated from the test models – also edited with Fujaba4Eclipse – which comprise IOD, SD, IS, and TD. In our approach, we use a *test table* representation of test scripts, which can then be automatically executed by FitNesse and Selenium.

Figure 4 shows an exemplary test table for scenario $<$*OpenAmazon*, *SearchBook*, *BuyItem*$>$. The generation works as follows: First, the algorithm finds paths in the IOD according to a coverage criterion (node, edge, or basic path coverage). Second, for every SD along a path, the algorithm creates a test table. This is done according to the Fixture definition from Sprint Planning 2, where it is defined how the rows are built for every keyword.

After the test tables are generated, the team can automatically run the tests. The test results show which test tables passed and which failed. When the implementation is fixed, all tests can be repeated very efficiently because they are automated.

Command	Target	Value
open	http://www.amazon.com	
type	twotabsearchtextbox	Sherlock Holmes
clickAndWait	//input[@alt='Go']	
clickAndWait	//img[@alt='Product Details']	
select	quantity	2
clickAndWait	submit.add-to-cart	
verifyTextPresent	quantity: 2	
verifyText	link=Sherlock Holmes	Sherlock Holmes

Figure 4: Exemplary test table for Selenium

The result of the sprint is an implementation that fulfills the selected product backlog ready for shipment. After this, the whole Scrum life cycle can start again.

4 Related Work

There is already some work on combining model-based testing with the agility. Katara and Kervinen [3] use a use-case driven approach, where their methodology is built around a domain-specific modeling language with action words and keywords. For behavioral modeling they use labeled transition systems (LTS), which enable a comprehensive specification of desired functionality. Using a coverage language, they control the test space. We believe that Scrum requires more user-friendly modeling notations than LTS and light-weight models in order to involve the customer into the process.

Winter [12] proposes to use Use Cases and Activity Diagrams for task modeling. The selected modeling languages are intuitive and can also be used for agility purposes. However, Winter does not address how to embed these languages into the agile development process.

Rumpe [6] proposes to use UML models and model transformation as centric concepts for agile processes especially for testing activities. However, the proposal is in an early stage and introduces no techniques how to handle the challenges in Scrum. In our paper, we propose concrete techniques for model-based testing. Our prototype shows that these techniques can be implemented with low effort.

5 Conclusion and Outlook

We have introduced a model-based approach for improving requirements specification and acceptance testing in Scrum. We use light-weight modeling notations of UML for specifying user stories from the beginning of the Scrum process. During Sprint planning, the user stories are enhanced with implementation details and test data. A prototype generates test tables, which can be automatically executed.

With our approach, we combine the principles of the *agile manifesto* with the techniques of *agile modeling* and *model-based testing*. We show that the introduced concepts can be supported by tools through the whole Scrum process.

The presented approach is in an early stage. Even if we believe that the modeling notations are easy-to-use, this assumption must be proven by an industrial case study together with customers. Also, the effectiveness of the prototype must be evaluated.

References

[1] A. Eberlein, F. Maurer, and F. Paetsch. Requirements Engineering and Agile Software Development. In *19th WETICE*, pages 308–313, 2003.

[2] B. Gloger. *Scrum - Produkte zuverlssig und schnell entwickeln.* Hanser Verlag, 2008.

[3] M. Katara and A. Kervinen. Making Model-Based Testing More Agile: A Use Case Driven Approach. In *Haifa Verification Conference*, pages 219–234, 2006.

[4] R. C. Martin, M. D. Martin, and P. Wilson-Welsh. FitNesse - Acceptance Testing Framework. fitnesse.org (last visited: 28.04.2010).

[5] OMG. UML 2.0 Superstructure Specification. Technical report, OMG, July 2005.

[6] B. Rumpe. Agile test-based modeling. In *Software Engineering Research and Practice*, pages 10–15, 2006.

[7] K. Schwaber and M. Beedle. *Agile Software Development with Scrum.* Prentice Hall, Upper Saddle River, 2002.

[8] ThoughtWorks. Seleniumhq - web application testing system. seleniumhq.org (last visited: 28.04.2010).

[9] University of Paderborn, Software Engineering Group. Fujaba4eclipse, September 2009. http://www.fujaba.de/projects/fujaba4eclipse.html.

[10] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann, 2007.

[11] VersionOne. State of Agile Development Survey 2009. Technical report, November 2009.

[12] M. Winter. SCORES: Objektorientierte Anforderungsermittlung mit der UML. In *Proc. of Workshop UML&Aufgabenmodellierung Mensch & Computer*, 2001.