Towards Model-based Acceptance Testing for Scrum

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

Abstract

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

Keywords

Agility, Scrum, Acceptance Testing, Test Automation, Model-based Testing

1 Motivation

Over the last years, agile methods have become more popular. Several methods like Scrum, Feature Driven Development (FDD) [3], and Extreme Programming (XP), 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 [2]. They all value the agile manifesto [1] and share some common principles: Improved customer satisfaction, adopting to changing requirements, frequently delivering working software, and close collaboration of business people and developers [2].

At the moment, Scrum [10] is the most used method; about 50 percent of business companies are using the method itself or a hybrid of Scrum and XP [14]. The main focus in Scrum is on the 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* [10]. This leads to poor requirements specifications, i.e. requirements are incomplete and inconsistent, which causes 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 implemented during the sprints. Thereby, integration testing has to be conducted in order to check the correct interaction of the new functions with the old ones. Additionally, regression testing has to be conducted for checking whether the integration of new functions has corrupted the old functions. Depending on the number of sprints and the functions to be integrated, the 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 propose a model-based technique for coping with the challenges addressed above. First, we want to improve the communication and the documentation of customer requirements by using simple UML [8] 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 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 [7] or Selenium [11], 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 [6, 13]: (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.

The idea of using models in agile development is not new. Scott Ambler introduced *Agile Modeling*, which is a collection of best practices and principles. He states that agile modeling needs *simple* but *multiple* models one of which fulfills a *concrete purpose*.

Our paper is structured as follows: next section gives a brief overview on the activities and artifacts in Scrum. Section 3 explains our model-based testing approach for acceptance testing in detail and gives an example how test scripts can be automatically generated from models. After addressing some related work in section 4, we give in section 5 an outlook for further research in this area.

2 Scrum

Scrum was invented by Jeff Sutherland, Ken Schwaber and Mike Beedle and is an empirical agile project management framework [10]. Scrum implies roles, artifacts and activities which are mainly meetings. Figure 1 shows the overall process of Scrum based on [4]. 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. This list is specified by the *product owner*, who creates, controls and manages the backlog. Originally, Scrum defines no special requirement engineering techniques for the creation of the product backlog but it is suggested to use so-called *user stories*[4].

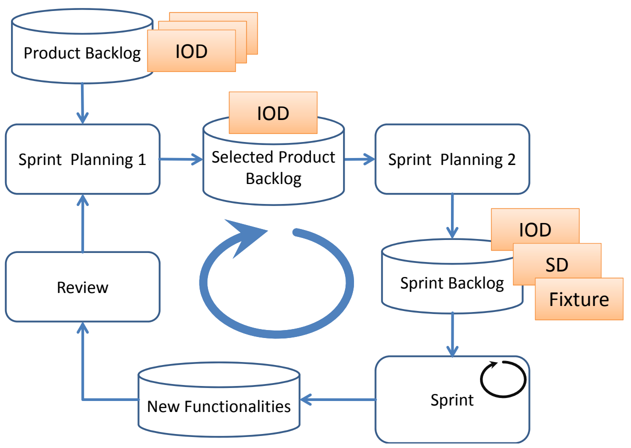


Figure 1: The Scrum process extended with model artifacts

After creating the product backlog, the main Scrum flow starts. The product owner and the *Scrum team* 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 in 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 are done (regression testing); (3) the integrated system of new and old functionalities have 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 [8] where the required functionalities are specified as *user stories*. 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* [7] which is a domain-specific language for specifying the interface between the user and the software system.

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 gets 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. Tester extends the SDs with *Test Data* (TD). All model artifacts are used for automatic test case generation resulting in test scripts which 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.

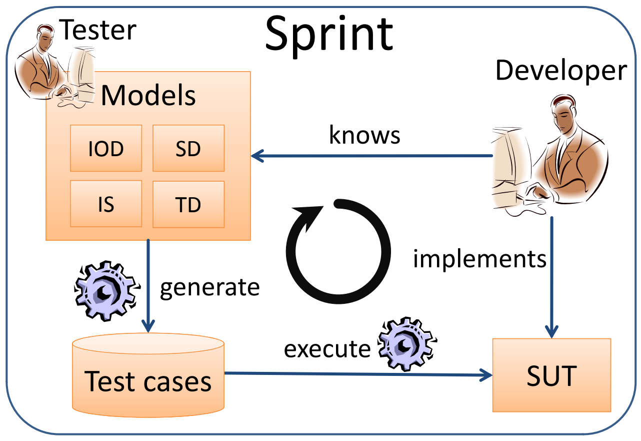


Figure 2: MBT activities embedded into the Scrum sprint

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 activity is given a name that describes the activity 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 give a good hint on the complexity of the user stories. If the IODs are too big to be realized in one sprint, they can simply choose a subset of paths from the starting to the ending node, leaving out additional branches. In our example, they could choose the path *OpenAmazonBookLink, BuyItem, Login, Confirmation* for the first sprint and *OpenAmazon, SearchBook, NavigateToBook* for following sprints.

The second Sprint Planning is used to refine the selected Product Backlog Items. In our approach, this means the team refines the activities in the IOD by adding SDs to them. In Figure 3.b, the SD for *BuyItem* is shown. It describes what interaction happens when conducting this activity, that is the user selects the quantity and clicks on the button to add the item to the shopping cart. 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 [11], which 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 where 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 our example shown 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 contains 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 also using the Fixture language. The verification step in Figure 3.c checks whether the quantity given by the user does appear correctly in the shopping cart.

For automating test execution, we have developed a prototype test case generator as a plugin for Fujaba4Eclipse [12]. 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 test tools like FitNesse or 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 (activity, 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.

For the Fixture keyword *select*, the according message is generated to the following row:| *select* | *quantity* | *2* |. The first column is the message name conforming to the used Fixture, the second the IS annotation, and the third a value from the TD table. The methods on the system lifelines (*show(ShoppingCart)* on *Browser*) specify the expected behavior of the sys-

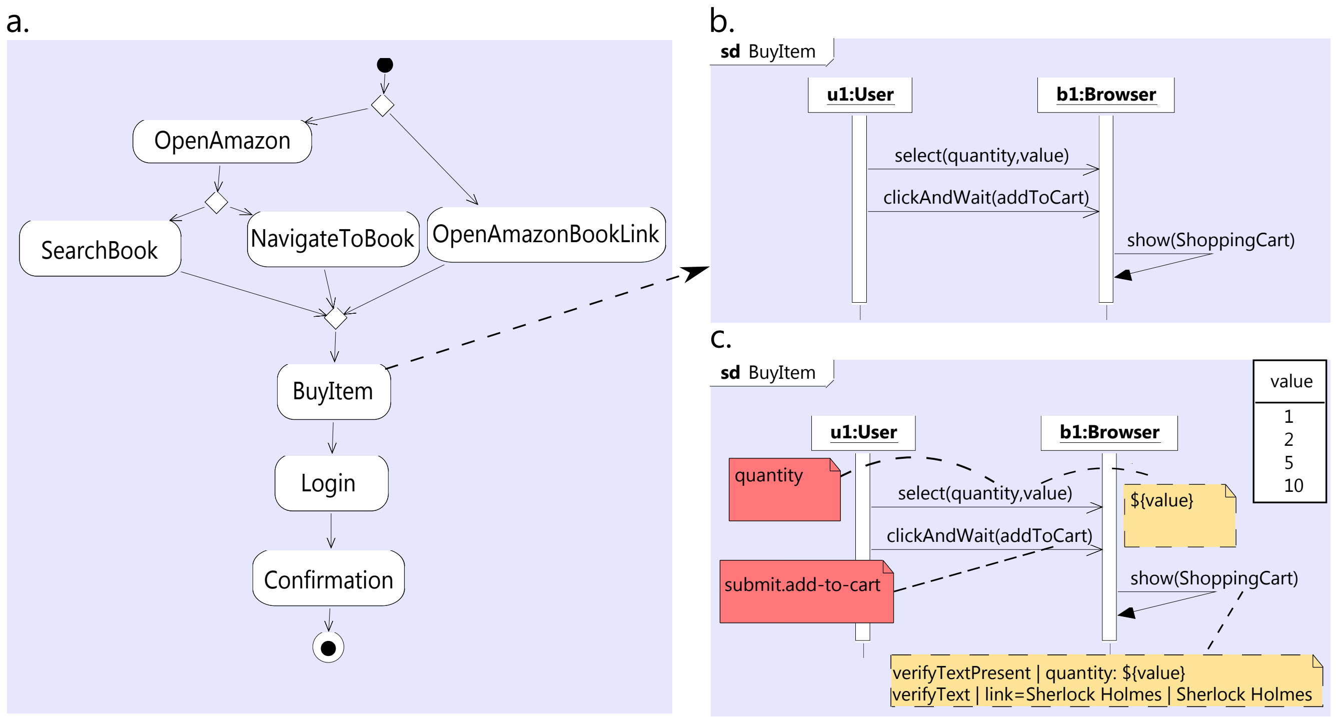
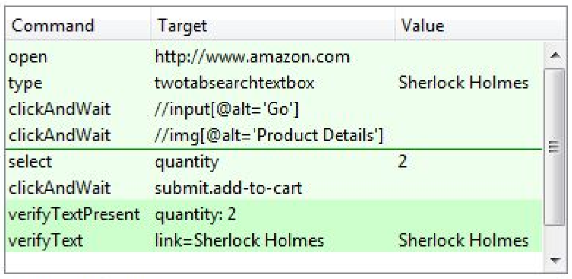


Figure 3: a. Interaction Overview Diagram describing a user story. b. Every activity is later linked to a Sequence Diagram. c. Sequence Diagram with annotations.



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

tem which is enhanced by the tester with some verification steps. Here, the algorithm generated two rows, one of which is | *verifyTextPresent* | *quantity: 2* |. In our example, we check that the quantity given by the user is correctly displayed in the shopping cart. We use facilities of Selenium to parameterize test tables thus iterating test execution for each entry in TD.

When 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.

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 [5] 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.

Rumpe [9] proposes using UML models and model transformation as centric concepts for agile processes especially for testing activities. However, the proposal is in a very early stage and introduces no concrete techniques how to handle the challenges of Scrum. The authors say "Neither are the tools ready for major practical use, nor are semantically useful transformations understood in all their details. Neither the pragmatic methodology, nor the underpinning theory are very well explored yet" [9]. In our paper, we propose very concrete techniques for model-based testing. Our prototype shows that these techniques are realistic and applicable.

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 thus serving as a specification for developers. Testers enhance them with test data and generate automatically test tables using a prototype test generator. Test tables can be then executed by Selenium which is a well-established tool for acceptance testing.

With our approach, we combine the principles of the *agile manifesto* with the techniques of *agile modeling* and *model-based testing*. Our prototype shows 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 for the product owner and the customer, this assumption must be proven by an industrial case study. Thus, the efficiency and the effectiveness of the approach must be evaluated. Also, the integration of the test generator into existing tool landscape must be evaluated.

References

[1] Various authors. Manifesto for Agile Software Development. http://agilemanifesto.org/ (last visited: 28.04.2010, 2001.

[2] Armin Eberlein, Frank Maurer, and Frauke Paetsch. Requirements Engineering and Agile Software Development. In *12th IEEE International Workshops on Enabling Technologies (WETICE 2003)*, pages 308–313, 2003.

[3] John M. Felsing and Stephen R. Palmer. *A Practical Guide to the Feature-Driven Development.* Prentice Hall International, 2002.

[4] Boris Gloger. *Scrum.* Hanser Verlag, 2008.

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

[6] Bruno Legeard. MBT for Large-Scale Enterprise Information Systems Challenges and Quality Issue. In *Key-note AMOST&QuoMBaT@ICST2010*, April 6th, 2010.

[7] Robert C. Martin, Micah D. Martin, and Patrick Wilson-Welsh. FitNesse - Acceptance Testing Framework. http://fitnesse.org/ (last visited: 28.04.2010, 2008.

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

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

[10] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum.* Prentice Hall, Upper Saddle River, 2002.

[11] ThoughtWorks. Seleniumhq - web application testing system. http://seleniumhq.org/ (last visited: 28.04.2010, 2004.

[12] University of Paderborn, Software Engineering Group. Fujaba4eclipse, September 2009. http://wwwcs.uni-paderborn.de/cs/fujaba/projects/eclipse/index.html.

[13] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann, 2007.

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