(INVITED TALK)

AGILITY VS. MODEL-BASED TESTING: A FAIR PLAY?

Baris Güldali, Michael Mlynarski

Software Quality Lab (s-lab), University of Paderborn
Warburger Str. 100, 33098 Paderborn/Germany
{bguldali,mmlynarski}@s-lab.upb.de

ABSTRACT

Agile manifesto defines principles for a light-weight software development process aiming at an improved customer satisfaction. *Automated testing* plays an important role in fulfilling these principles, because it enables efficient execution of *test scripts* for checking the quality of delivered software. However, the implementation and the maintenance of the test scripts can be very tedious and error-prone. In order to deal with that, *model-based testing* extends the automated test execution by test design and test implementation. Thus, model-based testing can speed up the test automation and improve the maintenance of test scripts. Nevertheless, introducing model-based testing requires some initial and some continual efforts, like creating test models, buying or developing tools, etc. In this talk, we will discuss how model-based testing can support agile development without conflicting with the principles of agile manifesto.

Index Terms - Agile manifesto, Automated testing, Model-based Testing

1. INTRODUCTION

As the complexity of software rises, novel software development techniques are required in order to cope with the technical and the organizational challenges in the development process. Model-based software development (MBSD) proposes using *abstract models* for better communication, for maintainable software specification and for efficient code generation. In this context, model-based testing (MBT) proposes using models for automating some of the testing activities, e.g. test case generation, evaluation of test results, which are tedious and error-prone tasks if they are manually done. In order to profit from model-based techniques in development process, however, some efforts must be expended, e.g. for introducing tools, for training developers and testers, for creating and maintaining models, etc. That is why MBSD is said to be a "heavyweight" technique for creating better software.

In contrast, agile manifesto [1] proposes a "light-weight" development process where (1) individuals and interactions are favored over processes and tools, (2) working software is favored over comprehensive documentation, (3) customer collaboration is favored over contract negotiation and (4) responding to change is favored over following a plan [1]. However, in the practice, these principles are likely to be misinterpreted such that developers often neglect documenting customer requirements properly. Frequently, this leads to chaos in the development process and to conflicts during the delivery and acceptance. Thus, it is a challenge to follow the principles of agile manifesto and thereby not to lose sight of the proper documentation and communication of customer needs and of the efficient and effective development.

We believe that, model-based techniques can help in dealing with these challenges. In the rest of paper, we will discuss how agility and model-based paradigm fits together. Thereby, we will mainly focus on the integration of model-based testing in agile development process as an enabling technology for the principles of the agile manifesto.

2. AGILE MANIFESTO

In 2001 seventeen software experts, who have introduced well-known agile methods (e.g. Scrum, Test driven Development (TDD), Extreme Programming (XP) etc.) have defined common principles for a lightweight development process. The new development paradigm should be an alternative to documentation-driven, heavyweight software development processes. They called these principles "agile manifesto". Agile manifesto includes the following principles (based on [1]):

1. *Customer satisfaction*: The highest priority in agile development has the customer satisfaction, which can be achieved by early and continuous delivery of valuable software. This principle has the highest priority in agile manifesto. All other principles serve to achieve this goal.
2. *Fast adaptation*: In agile development, requirements changes of the customer are

welcome, even in the late phases of the development. The flexibility in agile processes enables changes in software for assuring the customer's competitive advantage.

3. *Frequent delivery*: For customer satisfaction, it is important to show that the development process makes progress. For showing this to the customer, deliver new versions of software frequently. Define together with the customer what "frequent" means. The time slots can range from a couple of weeks to a couple of months. Try to keep the time slots as short as possible, because frequent delivery leads to frequent feedback.
4. *Close collaboration*: For achieving fast adaptation and frequent delivery, it is important to understand customer's business needs and consider them during the development continuously. For that, business people and developers must work together every day throughout the project.
5. *Motivated members*: Identify motivated team members who can push on the project. Provide them with the resources they need and support them while getting the job done.
6. *Conversation*: For achieving fast adaptation and frequent delivery, besides close collaboration with the customer, also the efficient communication between team members is important. The most efficient and effective method of exchanging information is face-to-face conversation.
7. *Working software*: Supply the customer with working software which is the main measure of progress. Delivering working software is indispensible for customer satisfaction.
8. *Sustainable development*: Agile processes promote sustainable development.
9. *Constant pace*: The customers and developers should be able to keep a constant pace for the whole time of project.
10. *Good design*: Continuous awareness for technical quality and good design improves agility.
11. *Simplicity*: Simplicity is crucial, which means that the amount of work to be done should be kept minimal.
12. *Self-organization*: Motivate team members to organize themselves.
13. *Reflection*: Motivate team members to reflect their experiences at regular intervals. Team members should discuss on how to improve the effectiveness and the efficiency in team and should suggest improvements accordingly.

Existing agile methods aim at enabling these principles. For example, Scrum promotes the close collaboration of customer or product owner at identifying software functionalities to be implemented in the next development cycles [4]. TDD advocates continuing programming until all predefined test cases are passed [1]. Since test cases are seen as specification, the resulting software is assumed to be correct with respect to the specification. Test automation plays in important role in agile methods supporting an efficient and effective development process. Having different focus, agile methods mostly should be combined in order to fulfill all principles of agile manifesto.

3. MODEL-BASED TESTING VS. AGILITY

We believe that model-based techniques can help in combining the different tasks in agile development by using abstract models as primary development artifacts. Models can support *communication* between team members and customers, *documentation* of customer requirements and design decisions and *automation* of code generation and testing. Thus, model-based techniques can enable an integrated development throughout the whole project. As next, we want to focus on how the documentation of customer requirements and their validation can be supported by model-based testing while following principles of agile manifesto.

3.1. Model-based Testing

With the emerging popularity of model-based software development, the usage of models in software testing is also desired. There are several definitions of model-based testing (MBT) in the literature, but the common understanding is that MBT is "the automation of test design of black-box tests" [2]. Therefore, MBT uses abstract models (test models) of the system under test (SUT) or its environment as the source for test generation. In addition to models of SUT and the environment, also the testware itself can be modeled [3].

There are three main advantages of MBT, which make this technique interesting: a) enabling high coverage, b) need for lower effort and c) enabling early testing. Because MBT uses sophisticated algorithms and tools for automatic test generation, far more test cases than while manual testing can be generated. This way a very high coverage of the system specification and/or requirements can be reached. While test cases are not designed and implemented manually anymore, the effort for this task is significantly low. This works under the assumption that the modeling effort is lower than the manual test design activity. Last but not least the early creation of test models supports the validation of requirements even before the system is implemented.

3.2. MBT as a technical enabler for Agility

Using MBT, the requirements can be captured and communicated in form of models. Unified Modeling Language (UML) provides many types of visual diagrams for describing the desired structure and behavior of software. Most of the diagrams have a quite simple syntax and fairly clear semantics such that customer and developer can easily learn how to

express their requirements more precisely, thus enabling the principle *close collaboration*. The changes in requirements can easily be made on the already created models, thus improving *fast adaptation*. Models can also support the *conversation* between team members, where the results of a discussion can be edited into the models immediately. Also the *simplicity* principle can be supported by models by using the abstraction, modularization and decomposition features of modeling.

There are different scenarios for creating and using models in MBT [9]. While some scenarios propose sharing models (one model for test team and development team), some scenarios require separated models (one models for each test and development team respectively). Using shared models can support *close collaboration,* face-to-face *conversation* and *simplicity*. However, if same models are used for development and testing, specification errors cannot be found [9]. Using separate models makes the teams for development and test more independent and enables finding specification errors, thus assuring *working software*.

Models having a well-defined syntax and semantics can be handled by computers, which obviously bring efficiency into the test process. The state-of-the-art modeling techniques support creating *good design*. Depending on the context of development, formal or semi-formal notations can be used. The more formal the models are, the better automatable are the test activities. Especially the automation of the test design task, which is the most costly and time consuming part in testing [5], leads to more efficiency. Test automation is the key for assuring *working software, frequent delivery*, *sustainable development* and *constant pace*.

Within MBT several coverage criteria for selecting test cases can be used. One possibility is to cover the customer requirements, which directly correlates with several agile principles. The *customer satisfaction* and *close collaboration* principles are supported by refining and understanding customer requirements while modeling them and showing that those requirements were successfully tested. The usage of different selection criteria and possibly combining them leads to higher defect detection rate and therefore facilitates *working software*. Due to changeable coverage criteria and automated test case generation, the test team can conduct different testing scenarios and gain experience for further development cycles and projects. This flexibility and configurability of MBT enables *reflection* in agile development.

4. A FAIR PLAY?

As discussed in the last section, MBT can definitely enable many principles of the agile manifesto. The main advantage of MBT for the agile world is the usage of models as primary artifacts and the automation of several test activities. This way MBT fits very well with agility!

However, MBT is not for free. Introducing MBT into the agile development process requires some initial and continual efforts as discussed in [6] and [7]. These include:

- Training team members for modeling
- Buying or developing modeling tools
- Buying or developing test drivers and test adapters
- Defining modeling notations and test selection criteria
- Creating and maintaining models
- Eventually extending generated test cases by test data
- Eventually evaluation of test results

At first sight, these efforts seem to be not proportional to the lightweight development purposes of agile manifesto. However, test automation is an indispensible part of agility enabling the efficient and effective process. Fewster and Graham said in 1999 that "automating chaos just gives faster chaos". MBT is an attempt to make test automation more systematic, more maintainable.

In this paper, we have discussed how agility and MBT conceptually fits together. A concrete approach for combining agility and MBT can be read in [8]. There, we have described a concrete approach including tool support for integrating MBT into Scrum.

5. REFERENCES

[1] Beck, K. et al.: Manifesto for Agile Software Development. Online resource at agilemanifesto.org (Last visited: 29.07.2010)

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

[3] Baker, P. et al.: Model-Driven Testing: Using the UML Testing Profile, Springer Verlag, 2008

[4] Schwaber, K., and Beedle, M.: Agile Software Development with Scrum, Prentice Hall, 2002.

[5] Pol, M. and Koomen, T. and Spillner, A.: Management und Optimierung des Testprozesses, dpunkt.verlag, 2002

[6] Güldali, B. and Jungmayr, S. and Mlynarski, M. and Neumann, S. and Winter, M.: Starthilfe für modellbasiertes Testen. OBJEKTspektrum, 2010, 3, 63-69

[7] Güldali, B. and Mlynarski, M. and Sancar, Y.: Effort Comparison of Model-based Testing Scenarios. Proc. of Quombat Workshop at ICST, 2010

[8] Löffler, R., Güldali, B., Geisen, S.: Towards Model-based Acceptance Testing for Scrum. Softwaretechnik-Trends, GI, 2010 (to be published)

[9] Pretschner, A., Philips, J.: Methodological Issues in Model-Based Testing. In M. Broy, et.al. (Eds.), Model-Based Testing of Reactive Systems, LNCS no. 3472, Springer-Verlag, 2005, pp. 281-291.

[10] Beck, K. Test-Driven Development: By Example. Addison-Wesley Longman, 2003