7

September, 2009

testing experience

The Magazine for Professional Testers

printed in Germany
print version 8,00 €
free digital version
www.testingexperience.com
ISSN 1866-5705

Agile Testing

© iStockphoto/Mlenny

Automated Integration Testing in Agile Environments

by Slobodanka Sersik & Dr. Gerald Schröder

The agile approach in software projects is not compatible with most of the established quality assurance processes. Quality assurance traditionally requires a finished product that must be verified against a finished specification. In agile projects, however, where response to change is more valuable than a fixed specification, a moving target must be verified against changing circumstances. Yet, testers are not able to develop test plans or automate tests more than one iteration in advance. Thus, automation of integration tests in agile environments is a difficult task. On the other hand the agile process supposes that testing happens closer to the developers in space and in time. Therefore, if the test automation effort is distributed among both developers and testers and if the test automation complexity is decreased through modularization and abstraction of reusable test components, thorough integration testing can be accomplished.

A large and growing variety of tools support automation of integration tests. Yet, most of them rely on GUI scripting by simulating users. But how can we automate integration tests for systems that

- are highly automated themselves and do not offer user interfaces?
- have use cases which are triggered by external systems?
- interact with external systems which cannot be included in the manual testing?
- offer several different user interfaces, and yet use a common back-end system?

In this article we will present a testing model we designed to specifically address these issues.

For better illustration of the problem and afterwards its solution, let's consider a simplified order and stock management system. It contains multiple front-ends: (1) an ordering user interface offered as web interface used by the consumers, and (2) a stock management desktop application used by the shipping department. Additionally, the system is dependent on an external system – the bank that actively sends bank transfers.

One typical story in the system under test (SuT) that defines a standard test case is presented in Figure 1, and can be explained through the following activities:

1. Customer places order containing goods and quantities interactively via web front-end
2. Order management system generates order reference number presented to customer
3. Customer initiates bank transfer giving order reference number
4. Bank sends bank transfers to order management system
5. Order management system matches unpaid orders against bank transfers using order reference numbers
6. For each paid order, a shipment order is being presented to the stock manager in his desktop application

How can we test this system efficiently and effectively? It can be done following a simple, nevertheless powerful model that differentiates four test development areas: (1) describing test scenarios spanning different system components, (2) implementing reusable test steps to create different test scenarios, (3) building simulators for machines or systems that cannot be integrated in the automatic integration test, (4) implementing adapters that allow a test scenario execution engine of an integration testing tool to control the different system components (or simulators).

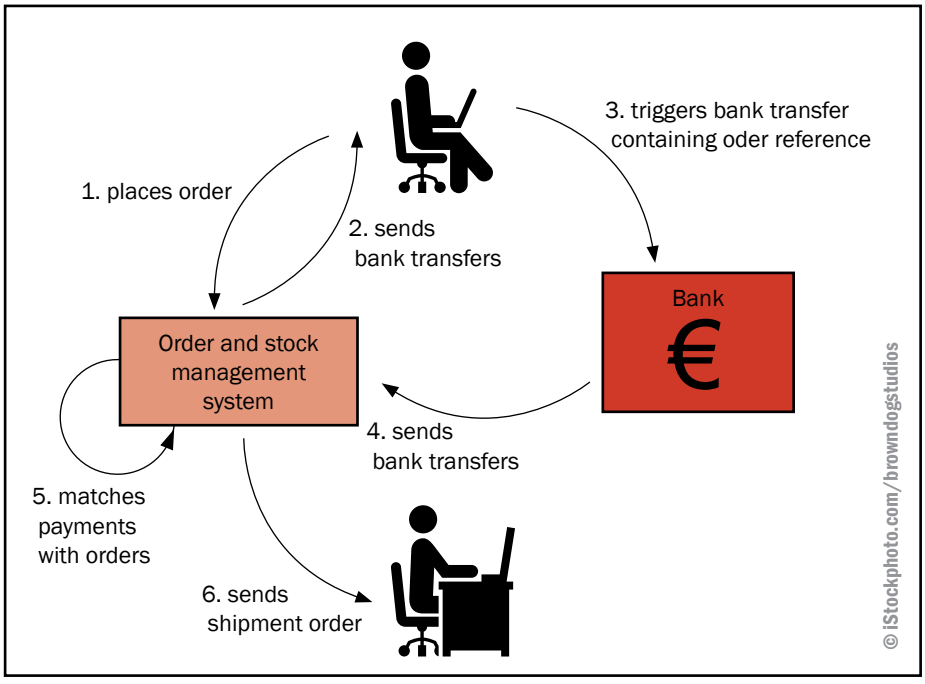

Figure 1: System under Test - Order and stock management system

Describing test scenarios spanning different system components

To distribute the testing complexity across different roles, domain testers and developers, the test description should be separated from test automation. In this way the domain testers can assemble and modify test scenarios using already available test steps implemented by the developers.

The goal of integration tests is to check the correct interaction between system components that have usually been coded by different people. Therefore testers who know and understand the overall system should prepare test scenarios spanning different system components and modularize them in test steps. The system developers themselves or test coders can implement these test steps.

In the order and stock management system described above the test scenario might be modularized as follows: place order, trigger bank transfer and read shipment order.

Implementing reusable test steps that are used to create different test scenarios

Integration tests need parameterized reusability. Why? Integration tests consist of different test scenarios that contain the same tests steps but differ in their context. For example, the placement of orders differs in the goods ordered. We do not want to implement the ordering process (list all goods, select a good from list, enter quantity, add to shopping basket, select next good, ...) for each integration test scenario that involves ordering of goods. So we create a reusable test step „place order" that is parameterized by the goods and quantities as test data. This step may be reused as a step in different test scenarios, parameterized with different test data.

Additionally, this method follows the DRY (Don't Repeat Yourself) principle that suggests a single point of maintenance. Consequently, technical changes such as new security query while ordering will be maintained in one place only, and not in each test case.

Building simulators for machines or systems that cannot be integrated in the automatic integration test

Simulators in integration testing are not just behavioral mocks (i.e., reacting to external stimuli); they have to be controlled explicitly depending on the test scenario. For example, the order system presents to the customer via its web interface an order reference number just generated to be used for payment. The integration test engine has to supply this order reference number to the bank simulator so that it may send actively (i.e. without being pulled by the order management system) a bank transfer containing this order reference number (other test scenarios may contain a distorted order reference number or a wrong sum).

Implementing adapters that allow a test scenario execution engine to control the different system components (or simulators)

Integration tests have to be robust against technical changes. We therefore advise that adapters are used in order to "wrap" the interfaces to the system under test. For example: the order management system changes its back-end interface to select goods from RMI to SOAP. Do we have to fix each integration test scenario? Hopefully not. We have abstracted away the placement of orders in an adapter used in all test steps that select goods.

The simulators are also controlled through adapters by a test execution engine. The adapters trigger active behavior and inject data into the simulators that would have been supplied by humans. Using adapters is convenient if the simulator is being replaced by another simulator or even the real system. In that case we only need to modify or exchange the adapter instead of the test steps.

Conclusion

Considering these four recommendations, the integration test for the depicted example – order and stock management system – is shown in Figure 2. Using the presented model we can build flexible and modularized tests that fully comply with the requirements of an agile project environment. We developed the model in various projects that focused on highly automated processes. Our experience showed that our approach provides an effective and cost-efficient way to build, maintain, execute and analyze automatic software tests. The introduction of this model in an agile project is a win for all four parties:

- Win for testing team: faster test automation and extreme flexibility on changes
- Win for developer team: prompt feed-

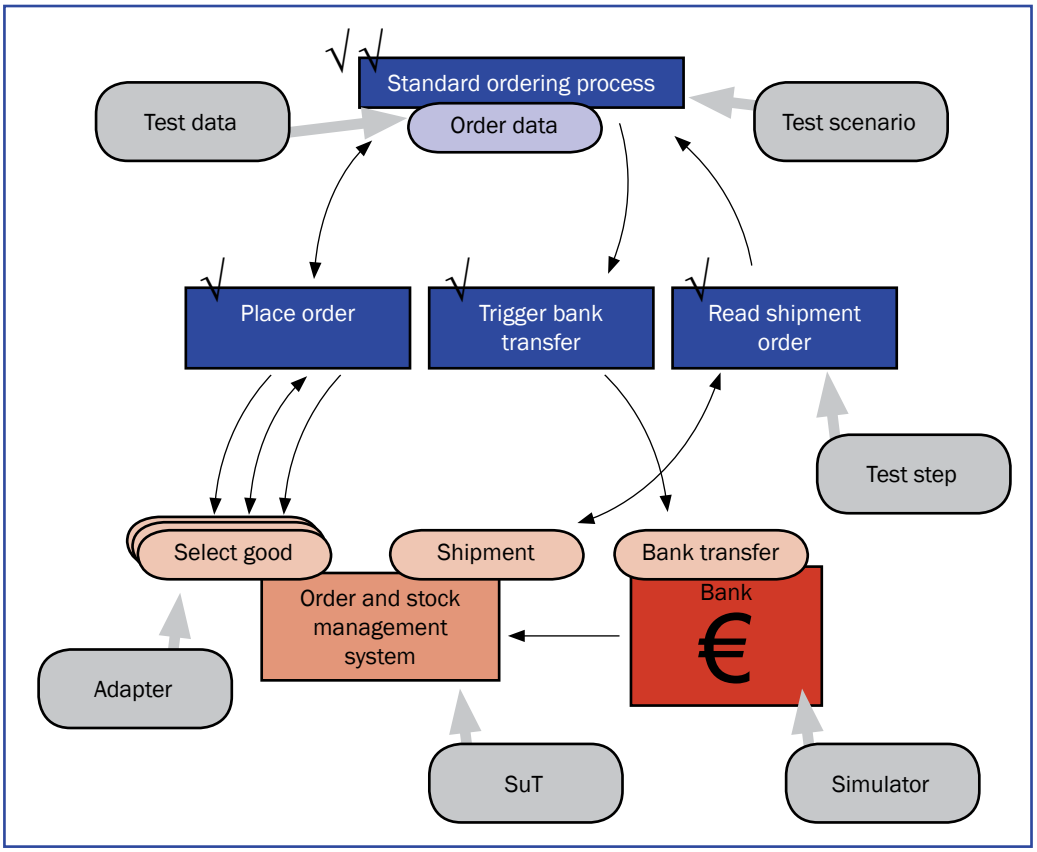

Figure 2: Integration Test - Order and stock management system

back about software quality and higher appreciation of testing efforts due to own contribution

- Win for management: lower costs due to on time failure detection and faster test automation
- Win for customers: on time delivery, high quality system

The integration testing model is implemented in an automated testing framework: the open source project iValidator (ivalidator.org).

Biography

Slobodanka Sersik is Senior Software Developer and Consultant in InfoDesign OSD GmbH. Beside her engagement on customer projects as software developer and architect, her responsibilities focus mainly on automatisation of integration and system tests. She manages currently the further development of the Open-Source Testing Framework iValidator.

Dr. Gerald Schröder is Senior Software Developer and Consultant in InfoDesign OSD GmbH. The focus of his work is put on Software-Engineering and Software-Architectures in large Java projects. Beyond that he is one of the main architects and developers of the OpenSource Testing Framework iValidator.