e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 69–87, DOI 10.5277/e-Inf160104

Highly Automated Agile Testing Process: An Industrial Case Study

Jarosław Berłowski[a], Patryk Chruściel[a], Marcin Kasprzyk[a], Iwona Konaniec[a], Marian Jureczko[b]

[a] *NetworkedAssets Sp. z o. o.*

[b] *Faculty of Computer Science and Management, Wrocław University of Science and Technology*

marian.jureczko@pwr.edu.pl

Abstract

This paper presents a description of an agile testing process in a medium size software project that is developed using Scrum. The research methods used is the case study were as follows: surveys, quantifiable project data sources and qualitative project members opinions were used for data collection. Challenges related to the testing process regarding a complex project environment and unscheduled releases were identified. Based on the obtained results, we concluded that the described approach addresses well the aforementioned issues. Therefore, recommendations were made with regard to the employed principles of agility, specifically: continuous integration, responding to change, test automation and test driven development. Furthermore, an efficient testing environment that combines a number of test frameworks (e.g. JUnit, Selenium, Jersey Test) with custom-developed simulators is presented.

Keywords: software engineering, testing process, agile software development, case study

1. Introduction

Software testing is a very costly part of the software development process, it is sometimes estimated to make 50% of the whole development cost [1], [2]. It is one of the main activities (at least should be) in agile software development methods. Beck and Andres [3] claimed it to be a measure of project progress and the main mean of assuring software quality. On the other hand, applying agile software development methods significantly affects the testing process. The agile methods usually require to test early and to have the tests automated. In consequence, testing is not left to the final phase, but requires sustainable investments during the whole process (including after-release maintenance of automated test cases). Some of the authors even recommend to start with testing [4]. The agile testing has been studied for several years, however, there are still unanswered questions regarding scalability [5], the role of testers [6] or test automation [7].

The goal of this research is to extend the body of knowledge concerning agile testing by documenting a real life software testing process. This paper presents a case study of a medium-size software project with special factors that affects the aforementioned process, i.e. requests for unscheduled releases and high complexity of project environment. The project is a Java Enterprise system in the telecommunication domain and its main purpose is to ease network devices controlling and management. Thus, there is a number of features that concern integration and communication with other systems and devices. Functional tests of such features involve items that are outside of the tested system, but are necessary for a successful test execution. Therefore, the test environment is complex and its management can consume a considerable amount of resources, specifically in the case of automated tests, where all changes

that come from test execution should be verified and reverted in order to ensure tests repeatability. On the other hand there are many, unscheduled release requests that concern the newest version of the developed system. Those releases regard presentations for end users or deployments into an environment for acceptance tests. Nonetheless, the released version must satisfy company quality standards and must be 'potentially shippable' [8]. The unscheduled releases could be explained as releases in the middle of a Sprint (the project is developed using Scrum), that are announced one or two days in advance and are requested to contain some features from the current Sprint and, of course, all the features developed in previous Sprints. The unscheduled releases create challenges. It is critical to ensure that the 'old' functionality still works correctly and there are very limited resources for regression tests, since there are the 'in-progress' features that must be straightened up before the release. The solution is somewhat obvious – test automation. In order to support the unscheduled releases a complex set of automated regression tests must be present and frequently executed. In consequence, it is possible to evaluate the current version of software instantly and decide if it is 'potentially shippable'. The paper describes how the challenges are satisfied, in which ways they affected the testing process and what results were obtained.

The case study presents an insider perspective, i.e. it is conducted by members of the team that develops the project. Therefore, it is possible to get a really deep insight, but unfortunately the risk of biased observation or conclusion grows simultaneously. In consequence, the case study is focused on a quantitative approach, which is associated with lower risk of subjectivity instead of the qualitative one, which is usually employed in similar case studies. The study is designed according to guidelines given by Runeson and Höst [9].

A number of concepts is used in this paper. Let us present explanations of them in order to clarify possible ambiguities. The authors refer to **project size**. The concept is in fact the metric which Mall [10] identified as one of the two most popular and the simplest measures of project size, namely the number of Lines of Code (LOC). The project investigated here is described as **a medium-size software project**. The small, medium and large scale is not a precise term, it is very intuitive (e.g. more intuitive than LOC). The investigated project has been classified as a medium-size one, since it seems to be a bit smaller than projects described in [5] and [6] which are reported as large ones. The authors refer to **costs of development and test**. This should be considered as the amount of time committed by the development team during project related activities. One of the investigated aspects of the testing process is **availableness for unscheduled release**. A possibility of making an application release is evaluated using results of the functional tests (Selenium tests and REST tests) executed in a continuous integration system (100% success of unit tests is a prerequisite of the functional ones). The application is truly ready for release if 100% of the functional tests are passed. The application fits for a presentation in the case when at least 90% of the tests are passed. If more than 10% of the tests failed, no form of release shall be considered. The investigated testing process was assessed using the concept of **the level of agility** that evaluates the adoption of different agile principles using the concept of **the level of adoption**. Both aforementioned concepts are borrowed from [5], further details about them can also be found in section 2.3.

The rest of this paper is organised as follows. The next section presents the goal and study design as well as a detailed description of the study context. The obtained results are documented in section 3. In section 4, the threats to validity are discussed and in section 5, related work is summarized. Finally, the discussion and conclusions are given in section 6.

2. Goal and Study Design

2.1. Context

The context is described according to the guidelines given by Petersen and Wohlin [11]. It corresponds with the suggested description structure

as well as the recommended elements of the context.

2.1.1. Product

The investigated project regards the development of a system which is used in the management of telecommunication services delivery. The main functionalities of the product are:

- managing customer access devices,
- managing and configuring network delivery devices,
- managing resources of IP addresses from the public and private pools.

The following technologies are used in the software development:

- `GWT` – Google Web Toolkit is a development toolkit for building and optimizing complex browser-based applications (https://developers.google.com/web-toolkit/);
- `Spring` – a comprehensive programming and configuration platform for modern Java-based enterprise applications without unnecessary ties to specific deployment environments (http://www.springsource.org/);
- `Hibernate` – It is a framework which allows to provide the service of the data storage in a database, regardless of the database system (http://www.hibernate.org/);
- `Oracle DB` – Relational database management system which is commonly used in corporations because of data maintaining capabilities, security and reliability (http://www.oracle.com);

Maturity. The project was conducted from May 2011, the first release was carried out in Autumn 2011. The study was conducted in January 2013.
Quality. The main means of ensuring the product quality are tests. Further details regarding the testing process are presented in the next sections.
Size. The total number of Lines of Code is 327387 (calculated by StatSVN – all lines are counted).
System type. Enterprise system with a web UI. The development team makes use of functionalities accessible as REST services and also integrates a them with existing systems.
Customisation. It is custom software development. The product is tailored to the requirements of a customer (a telecommunication vendor).
Programming language. The system is developed mostly in Java.

2.1.2. Processes

Figure 1 presents the testing process. In fact, no defined process is used, and what the diagram shows is the result of the 'definition of done' that is being employed. New user stories are first identified, then acceptance criteria for them are defined and stories are estimated. When the Sprint begins, the development team makes the Sprint planning and later software development and unit tests. At the same time a specification for functional tests must be prepared. If pair programming was not carried out, than a code review is needed. Before the end of the Sprint, functional tests must be conducted. When there is time, automated functional tests are prepared, in other case, automation is postponed to the next Sprint. Test automation is not a part of the 'definition of done.' It might seem strange as automation is very important in agile processes, but unfortunately automation is very time-consuming and for technical reasons sometimes must be done after implementation (this remark does not regard unit tests). In consequence, it was not possible to always do implementation and automation in the same Sprint. Having the choice to either do longer Sprints or postpone automation it was preferable to postpone the automation as there is the requirement for unscheduled releases, which does not correspond well with long Sprints. Postponing test automation is not a good practice, but conducting functional tests manually [12, 13] or automating after the sprint [5,14] is not unique in software development. Puleio [14] even invented a name for not doing the test automation on time, i.e. testing debt. At the end of the Sprint the software is potentially shippable as the new features which are not covered by automated tests (if there are such) are tested manually. As presumably it already emerged from the above description, the employed development process is Scrum.

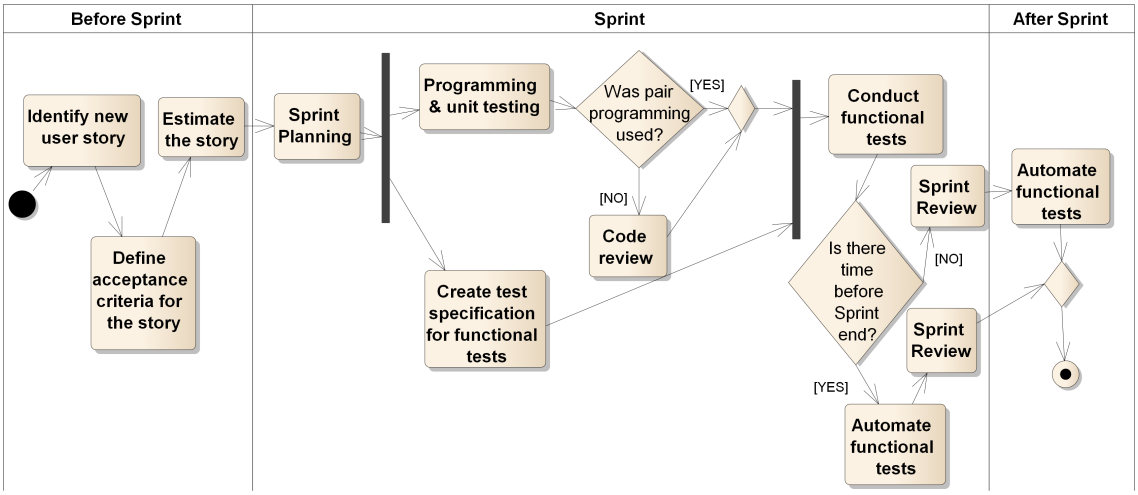


Figure 1. Testing process

2.1.3. Practices, Tools, Techniques

CASE tools

Eclipse STS (http://www.springsource.org/sts) is used as the primary development environment. Furthermore, the development team uses a set of frameworks for test automation which are employed in a continuous integration system (details in the next section).

Practices and techniques

- Time–boxing.
- Frequent (in fact continuous) integration.
- Frequent, small releases.
- Early feedback.
- Test automation.

2.1.4. People

Roles

- Product Owner – comes up with new ideas, updates priorities and chooses the most important issues, he is also responsible for contact with customers.
- Scrum Master – takes control of the Scrum process and enables team members to reach the goal of the Sprint by helping them with all impediments. For a short period of time there were two Scrum Masters, as the project was developed by two different teams.
- Team – responsible for reaching the goal of the Sprint. The development team currently consists of 8 people including: software developers, testers (one tester performs also the role of a business analyst), the Scrum Master (who takes part in software development) and the Product Owner (who does not take part in the software development):
  - Software developers (5),
  - Scrum Master (1),
  - Testers (2).

2.2. Experience

Some software developers have longiterm experience in the IT field, i.e. two people 5+ years of experience (one of them is a Scrum Master), the Product Owner 15+ years of experience and one of the testers who is also a business analyst 10+ years of experience. For the rest of the team, this is the first job. They take part in the project, learn technical skills and expand knowledge using different tools. The entire team has completed bachelor's or master's degree studies in computer science. In order to improve the qualifications, the members of the development team have participated in some trainings and certification programs, for example ISTQB (International Software Testing Qualifications Board) – both testers, Professional Scrum Master I – the Scrum Master, CISCO certificates (CCNA/CCNP, CCAI) – one of the testers.

The development process was configured by the experienced team members who also take care

of the introduction of their younger colleagues. Thus, there are no reasons to believe that there are issues in the process resulting a lack of experience. Additionally, the trainings and certification program were used to ensure high development standards and to avoid shortcomings.

2.3. The Level of Agility of the Investigated Testing Process

The paper is to be focused on agile testing process. Hence, it is important to take carefully analyze the principles of agility and assess the process. Otherwise, there would be a considerable risk of investigating other phenomena, not the ones that should be investigated.

2.3.1. Survey Design

The issue is addressed with a survey conducted among all the staff that is or was involved in project development. The survey is designed according to a set of weighted criteria suggested by Jureczko[1] [5], i.e. the following criteria are used (each of them can be met fully, partly or not at all):

- Test driven development – TDD is one of the most remarkable improvements, which has been brought to testing with the agile methods [15], [3]. Presumably not every agile testing process uses TDD, on the other hand TDD affects the process and system design in such a significant way, that it cannot be ignored when evaluating the level of agility (weight = 3).
- Test automation – most of the agile methods move the focus from manual to automated tests, e.g. [3]. Furthermore, the automation is often considered an essential practice [16], [14] (weight = 3).
- Continuous integration – what is the use of automated tests when they are not executed frequently? (weight = 2).
- Communication – it is considered at two levels, i.e. not only between team members but also with the customer, however, with respect to testing process the most relevant communication is between developers and testers (weight = 2).
- Pair programming – pair programming does not relate to testing directly. Nevertheless, it affects the quality of a source code – could be considered as on-the-fly code review (weight = 1).
- Root-cause analysis – it is one of the quality related eXtreme Programming corollary practices. It forces complex analysis for each of the identified defects. Not only the defect should be resolved, but also the software development process should be improved to avoid similar defects in future. The root-cause analysis is not recommended when some of the essential eXtreme Programming practices are not adopted. Therefore, there are agile testing processes that do not employ it (weight = 1).
- Working software (over comprehensive documentation) – one of the Manifesto for Agile Software Development rules that may have strong influence on tests (weight = 1).
- Responding to change (over following the plan) – another rule from the aforementioned manifesto. In the context of a testing process, this rule is mainly considered with respect to the flexibility of test plans (weight = 1).

The above listed criteria are extracted from principles suggested in eXtreme Programming [3] and Agile Manifesto. Each of them is somehow connected with testing and as the Authors believe they make a rich enough subset of all agile principles to offer a good understanding of the project reality with respect to the testing process. The questionnaire was generated in a paper form, nonetheless, it is available on-line: http://purl.org/MarianJureczko/TestingProcess/Survey.

2.3.2. Survey Results

The questionnaire was completed by all present team members and those past members that are still working for the company. The results are dis-

[1] The author argued the usage of weights by stating that the principles of agility have different relevancy from the testing perspective. It is also noteworthy that some of the agile software development methods, e.g. XP [3], identify categories of principles that are based on the relevancy of their adoption.

cussed in detail in the forthcoming subsections and presented in Figure 2. Each of the sub figures shows which respondents recognised given levels of adoption. It is worth mentioning that there is no single 'I do not know' response. Presumably it is a consequence of the fact that each of the respondents actively participates or participated in the investigated project.

Test driven development More than half of the respondents recognised test driven development as partly implemented and the rest of them as fully implemented (Fig. 2a). TDD has been used in the project for a long time, however, not all features are developed in this way. It is always the developer's responsibility to choose the most efficient way of the development and there are no repercussions for not doing TDD. Regardless of the selected development method, the unit tests are obligatory and hence TDD is often a good choice. Nonetheless, there is also a number of features that are closely coupled to moderately documented, external services which do not make a perfect environment for TDD.

Test automation Almost all respondents recognised test automation as fully adopted (Fig. 2b). test automation is very important in the investigated testing process. The automated unit tests are explicitly written in the project's 'Definition of Done' [8]. The automation of functional tests is also obligatory, however, sometimes it is postponed and it is not conducted in the very same sprint as the functionality which has to be tested. Typically, there is one automated test case per a user story, but there are also some very complex stories that are tested by more test cases and some complex test cases that cover more stories. All the automated tests cases are used as regression tests and are executed in a continuous integration system which brings us to the next evaluation criterion.

Continuous integration Each of the respondents acknowledged that the continuous integration principle was fully adopted (Fig. 2c). There is only one development line, and as far as it is possible the development team is avoiding using branches and promotes frequent commits. Moreover, there is a number of jobs defined in the Hudson (http://hudson-ci.org/) system to support automatic compilation, testing and deployment of the developed system.

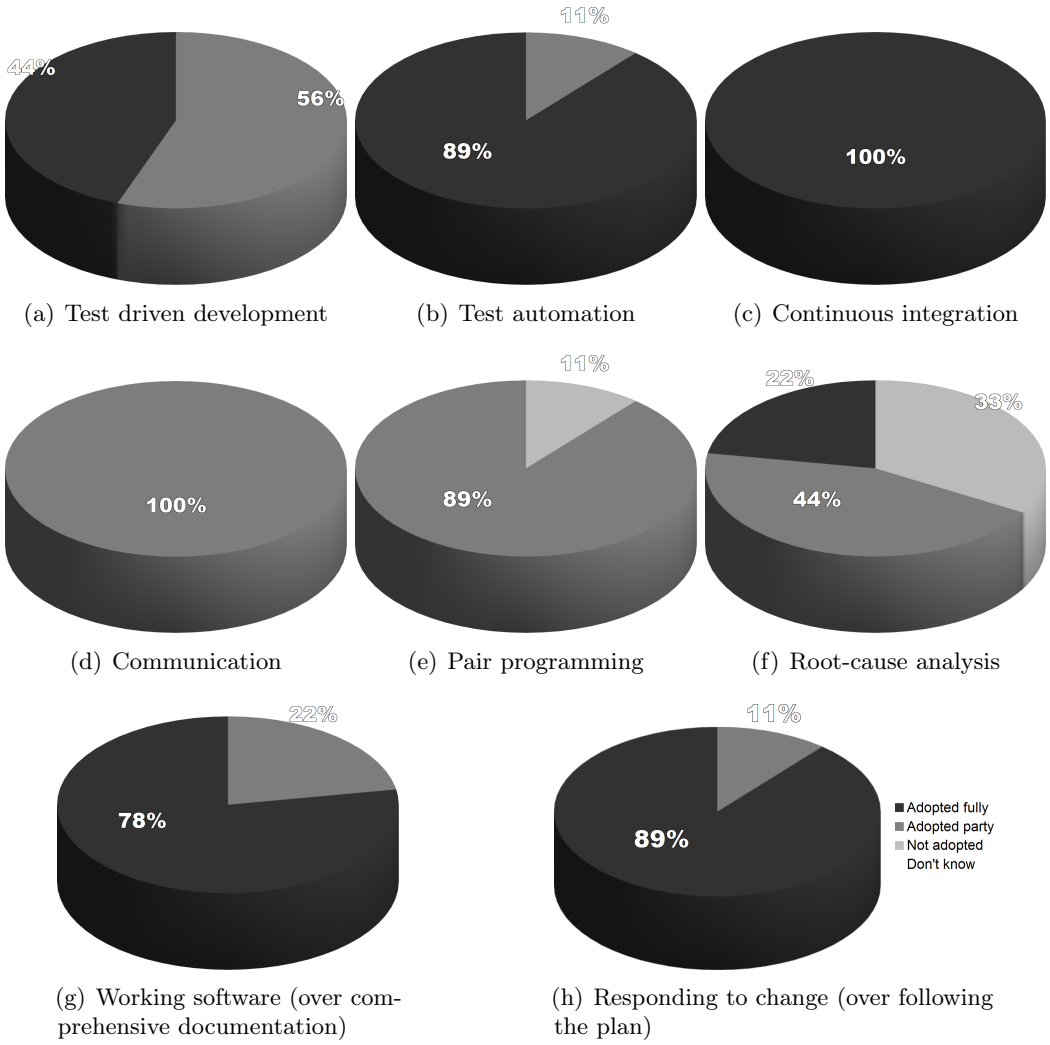
The unit tests and some of the functional tests are executed after each commit. The functional tests are split into two groups. The first of them contains test cases that do not need a great amount of time for execution, which in fact means no interaction with graphical interface – these tests are executed after each commit. The second group of tests contains GUI related tests and for performance reasons it is executed nightly.

The Hudson continuous integration system supports also releases. After a Sprint Review Meeting, i.e. when the sprint ends, a Hudson job is executed and performs the following actions:

- The version number of the developed system is updated.
- A SubVersion 'TAG' is created for the new version of the developed system.
- Release notes regarding newly implemented features are generated.
- A new version of the developed system is deployed to Apache Maven repository and saved on a FTP server.
- The new version is installed automatically in the customer acceptance tests environment.

Communication All respondents recognised the Communication principle as partly adopted (Fig. 2d). The communication was considered at two levels, namely among team members, specifically between testers and developers and between team members and the customer. The communication between team members is acknowledged to be effective. Testers and developers work in the same location. The communication is mostly verbal (but all major issues are reported in an issue tracking system) and supported by the Scrum meetings, e.g. Daily Scrum Standup Meeting. Furthermore, the roles (i.e. testers and developers) are not fixed, thus a team member has an opportunity to do developing as well as testing tasks.

The communication with customer was not assessed so well. The eXtreme Programming 'on site customer' principle [3] has not been installed. Furthermore, customer representatives do not participate in Planning and Review meetings [8]. Communication channels usually go through



(a) Test driven development (b) Test automation (c) Continuous integration

(d) Communication (e) Pair programming (f) Root-cause analysis

(g) Working software (over comprehensive documentation) (h) Responding to change (over following the plan)

Figure 2. The level of agility in testing process

Product Owner, which sometimes makes the communication inefficient.

Pair programming Most of the respondents recognised pair programming as partly adopted (Fig. 2e). The usage of pair programming in the project is limited – more often informal code reviews are used instead, usually in the form of a code walk-through. Nonetheless, presumably each team member experienced this practice since it is extensively used as a knowledge transfer tool. New employees work in pairs with the more experienced ones in order to improve their learning curve.

Root-cause analysis Presumably there is a confusion over what it means to adopt this principle since no clear message comes from the questionnaires (Fig. 2f). Definitely the root-cause analysis is not executed for all identified defects, only a fraction of them is so closely investigated. On the other hand, there are Sprint Retrospective Meetings [8] which address the most important issues and give the team an opportunity to identify remedies.

Working software (over comprehensive documentation) Most of the respondents acknowledged that the working software is valued over comprehensive documentation (Fig. 2g). As a matter of fact the customer is not interested in technical documentation. He is provided only with the user and installation guide. Therefore, the team could decide which technical documents to use and there are no reasons for preparing doc-

uments that are not useful. The Agile Modeling principles [17] are followed and in consequence the number of created documents is limited and always corresponds with one of two purposes, i.e. a model to communicate or a model to understand. The Product and Sprint Backlogs are used, but neither of these documents is delivered with the released product.

Responding to change (over following the plan) Most of the respondents recognised that responding to change is valued over following the plan (Fig. 2h). With regard to test plans the principle is fully adopted since test plans are never fixed and there is always room for an update. It looks slightly different in the case of the set of features (user stories) that are chosen for implementation during a Sprint Planning Meeting. The plan created during the aforementioned meeting shall not be changed according to Schwaber [8]. However, there is a possibility to terminate a Sprint and plan a new one (which has been done several times during the project). Furthermore, sprints are not long (for several months one week sprints were used, currently they have been extended to two weeks), thus waiting for a new sprint with an unplanned change is not painful. It should also also stressed that there is no fixed long term plan. Each new sprint is planned from scratch and hence unexpected changes can be easily handled.

Conclusion The results were evaluated in the way suggested in [5], i.e. value 1 was used for full adoption of an agile principle and 0.5 for partial adoption, then weighted average was calculated (the weights are given in teh previous subsection) and the value of 76.6% was obtained. The value is higher than the one calculated for the project investigated in [5], which could be interpreted as a slightly better adoption of agile testing principles in the described in this study process.

2.4. Objective

The study is focused on a testing process in a medium-size software project with challenging requirements for unscheduled releases and complex infrastructure to deal with. The paper presents how the testing process was tuned with respect to the aforementioned requirements, hence it can be considered as a descriptive or exploratory study [9]. The Authors believe that other practitioners who operate in a similar context will find this work helpful and use it as an example of a well working testing process. The study objective can be defined as follows:

Describe and evaluate an agile testing process with support for unscheduled releases in development of a software system that depends on a number of external services and devices.

The investigated testing process deals with two challenges. Rge first of them comes from business, i.e. there are often unexpected opportunities which must be addressed immediately, i.e. within one or two days depending on a release, otherwise they are missed. In consequence, sometimes it is not possible to wait with the release till the end of a Sprint. The second challenge comes from the project domain. The developed system operates in a complex environment that consists of a number of different network devices and services. Thus, it is a typical case when the configuration of the surroundings requires more time that the test execution itself. This challenge significantly affected the testing process, and therefore the Authors believe that it is a very important part of the project's big picture and must not be ignored in the case study.

2.5. Research Questions and Data Analysis Methods

RQ1: To what extent is the requirement for unscheduled releases satisfied? The unscheduled releases are one of the main drivers of the definition of testing process. Thus, it is critical to study this aspect. It will be evaluated using a quantitative approach. The continuous integration server will be employed as a data source and the results of the builds that execute functional tests (builds with unit tests are a prerequisite) will be used as a measure of the possibility of a release. In order to quantify the data we assumed the a release is possible when all tests are passed. Additionally, we assumed the possibility

of a release with acceptable risk of failure when not more than 10% of tests failed. Such a risk can be accepted only when the release is conducted exclusively for presentation purposes. The 10% threshold is provided to give insight into the variability of the continuous integration outcomes. It also corresponds with the business requirements as 90% of available functionality is usually enough to conduct a presentation. The Authors assumed that the release is possible when before 1 PM the build is successfully finished according to the aforementioned criteria. The time, i.e. 1 PM, was selected as it was the latest possible hour that enables installation in the customer environment or preparation of a presentation (depending on the goal of the unscheduled release) during the same working day.

RQ2: How is the test automation performed? Test automation is the main mean to address unscheduled releases as it is the only way to quickly assess the current software quality and decide if it is acceptable for a release. Furthermore, automation has a crucial role in the testing process and requires significant efforts. Specifically, in a project that operates in a complex environment that creates non-standard requirements for test configuration and in consequence out-of-the-box solutions are not sufficient. Hence, it is vital to describe in detail how the test automation is done. This research question will be addressed using the qualitative approach. All the employed testing frameworks, tools and home-made solutions will be described with respect to their role in the project.

RQ3: How much effort does the test automation require (with respect to different types of activities and tools)? Automated tests play a crucial role in deciding about an unscheduled release and since two different test frameworks were used it could be very interesting to evaluate the process from a business perspective as well. Hence, the study presents data regarding costs of test automation that allow to justify whether it is worth making the investments and which framework should be be chosen to have support for unscheduled releases. The company tracks data about committed efforts using the issue tracking system (Atlassian JIRA). Therefore, it is possible to address the third research question by mining the collected data and extracting information regarding efforts connected with creating new automated functional tests as well as with maintaining the existing ones. The results are presented using descriptive statistics and statistical tests.

3. Results

3.1. To what Extent is the Requirement for Unscheduled Releases Satisfied?

Research regarding availableness for an unscheduled release was conducted over a period of two months. The results are shown in the Figure 3. For 47.4% of the time, the application was ready for the release. A period of release with acceptable risk – 10,5%. The application was not ready for release for 42.1% of the time. The obtained results can be claimed as satisfactory with respect to availableness for an unscheduled release.

Kaner et al. [18] considered test coverage in the context of requirements-based testing, which is very similar to the REST and Selenium functional tests, as they are intended for proving that the program satisfies certain requirements. There is at least one automated test per requirement, which the investigated project is expressed using user stories. Therefore, there is 100% test coverage at the requirement level, i.e. each of the requirements is tested. However, it is an overoptimistic interpretation as not all corner cases in some of the user stories are automated and thus there are places where errors cannot be detected using the REST or Selenium functional tests. The tests execution results may also misleadingly show errors in the case of database malfunction or overload of a machine on which the test environment is located. Adding further tests would decrease the probability of releasing a low quality version of the system by limiting the number of not covered corner cases but it would also increase the probability of blocking a high quality release which can happen as a result of the aforementioned continuous integration

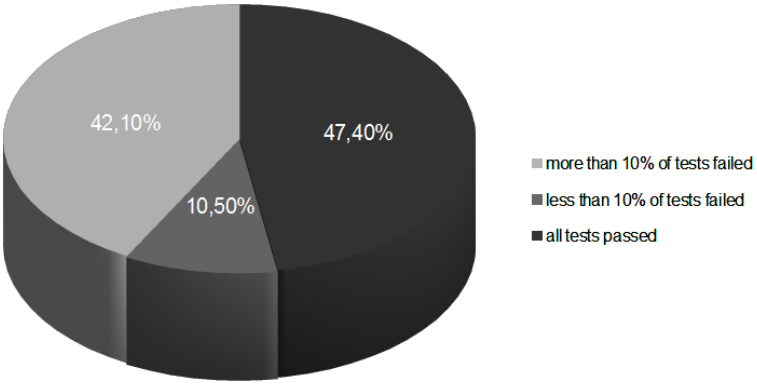


Figure 3. Results obtained from the continuous integration system

system malfunctions. Additional tests would also increase the costs of tests maintenance.

3.2. How is Test Automation Performed?

The results of the execution of automated tests are used as the primary citerion in making decisions regarding unscheduled releases. In order to ensure that tests results correspond with system quality, the team uses a wide variety of tools and frameworks. This includes also several self-developed solutions which contribute to conducting test automation in complex environments.

3.2.1. JUnit Test Framework

JUnit is a test framework for the Java language. On its own it provides the basic functionality for writing tests, which can be further enhanced by other extensions. JUnit tests also serve as an entry point to nearly every type of test in the project development (including the aforementioned self-developed solutions). It is a first class citizen with regard to the way the unit testing and test driven development are conducted. The unit tests are a part of the build, and thus the system cannot be built (and in consequence released) when some of the tests do not pass. There is also a positive side effect, i.e. developers are forced to instantly fix issues detected by unit tests.

EasyMock class extensions. EasyMock provides Mock Objects for interfaces and objects through class extension which is used to replace couplings to external systems or dependencies in unit tests.

3.2.2. Automated Selenium Tests

Automated regression tests with Selenium WebDriver are used to test whether high level functionalities and a graphical user interface work and react properly to input. These tests are combined into a suite and executed nightly within the continuous integration system (a new build of the system under test is deployed at the end of every day on a dedicated server and its database is set to a predefined state). These tests are not executed after each commit, as it is in the case of JUnit and REST tests, since, due to their complexity, the execution takes more than one hour. It is more than the recommended 10 minutes [3] and thus would have decreased the comfort of developers' work.

Tests using Selenium WebDriver are functional, they simulate a user action on the interface and check whether the output matches expectations. Typically a test covers exactly one user story, however, there are exceptions that span multiple test cases and user stories. Additionally, for tests where a connection to an external device is required (and it is not feasible to keep a real device connected to a test machine at all times), the team developed simulators which can be set up to respond like this particular device would (see 3.2.5). The development team strives to have at least one automatic functional test for every user story.

3.2.3. Jersey Test Framework

Tests prepared using Jersey Test Framework are referred to as the REST tests for short. The framework enables functional tests for REST services. It launches the service on an embedded container, sends HTTP requests and captures the responses.

The REST tests are used to test functionality of the system (each user story is covered by at least one Selenium or REST test). The communication between a client and a server and its validity is tested, unfortunately defects from a graphical user interface cannot be detected in such an approach. The REST tests are a part of the build and thus broken tests are early detected and fixed.

3.2.4. DbUnit

DbUnit is a JUnit extension targeted at database-driven tests that, among other things, puts a database into a known state between test runs. DbUnit is used for preparing database for tests. Specifically, it helps in creating the HSQLDB in-memory database that is used during the Selenium and REST tests.

3.2.5. Custom Developed Simulators

In some cases the system under test requires communication with a device or an external service. To solve this problem the team developed:

- SSH Simulator (https://github.com/NetworkedAssets/ssh-simulator/) to simulate communication with a device through the SSH protocol.
- Policy Server Simulator to test communication with a policy server through HTTP.
- WebService simulators to test WebService clients.

The simulators are used in the Selenium and REST tests.

SSH Simulator can be configured to read and respond to commands received through the SSH protocol. The exact behaviour, such as what response should be given to request, is configured in an XML settings file.

The SSH Simulator can be used in two ways. It is possible to launch it as an SSH server or as a temporary service for a single JUnit test. The SSH Simulator tool is configured using XML files. The XML configuration file contains the expected requests and responses that will be generated by the simulator:

SSH Simulator configuration file

```
<test_case ...>
  <login>login</login>
  <password>password</password>
  <device_type>CNR</device_type>
  <request delay_in_ms="3000">
    <request_command>dhcp reload</request_command>
    <response_message>
100 Ok
    </response_message>
    <response_prompt>nrcmd>$</response_prompt>
  </request>
</test_case>
```

The configuration file contains the `<test_case>` entry that represents the sequence of requests and responses mapped using series of `<request>` nodes (on the presented listing there is only one) that define the expected client requests and instruct the simulator how to reply to them. When the XML configuration file is ready, it is enough to use it in a JUnit test case as it is presented in the listing below:

Using SSH-Simulator in JUnit

```
public class MyTest extends
                  SshSimulatorGenericTestCase {
    @Test
    public void sampleTest() {
        initializeNewSshServer(xmlConfigFile,
                               ipAddress, port);
        //do the tests here
    }
}
```

The WebServices simulators are made using J2EE classes from `javax.jws` and `javax.xml.ws` packages. There is a dedicated class that is responsible for launching the WebServices in a separate thread.

Executing a simulated WebService for test purposes

```
public class WebServiceExecutor {
 private static Endpoint endpoint;
 private static ExecutorService executor;
 ...
 public WebServiceExecutor(NaWebService ws) {
   if (endpoint != null &&
                 endpoint.isPublished()) {
      endpoint.stop();
   }
   endpoint = Endpoint.create(ws);
 }
 /** Starts the web service */
 public void publish() {
   if (executor == null) {
     executor =
     Executors.newSingleThreadScheduledExecutor();
   }
   endpoint.setExecutor(executor);
   endpoint.publish(WS_URL);
 }
 /** Closes the web service and verifies
     execution results */
 public void shutdown() throws Throwable {
   endpoint.stop();
   assertTrue(webService.isOk());
   }
}
```

The web service has its own thread, but the 'shutdown' method is called from the JUnit context. Therefore, it is possible to assess what calls the received WebService. It is done by using the 'isOk' method which should be implemented by each of the simulated WebServices:

Example of a WebService implementation

```
@WebService(name = "WS",
                     serviceName = "WSService")
public class MyWs extends NaWebService
   ...
   @WebMethod(operationName = "OP",
                              action = "urn#OP")
   @WebResult(name = "Result",
                 targetNamespace = "http://...")
   @RequestWrapper(...)
   @ResponseWrapper(...)
   public Result op(@WebParam(name = "NAME"...)
                                 String name...)
         throws OperationFault_Exception {
      try {
         assertEquals("PCSCF", name);
      } catch (Throwable t) {
         log.error(t.getMessage(), t);
         reportError(t);
      }
      return = new GenericResponseType.Result();
   }
}
```

The presented example shows how to test the value of a WebService parameter. WebService simulators are usually employed in the REST tests and are useful in testing WebService clients.

3.3. How Much Effort Does the Test Automation Require (with Respect to Different Types of Activities and Tools)?

To answer this question, it is necessary to refer to historical data about the efforts made by team members to create and maintain existing automated tests. For this purpose, data from the issue tracking system used by the company (Atlassian JIRA) were collected.

Then the data about all automated tests have been divided into two categories according to the used tool: Selenium tests and REST tests. This distinction has been made due to the fact that these tools tests different layers of the software. The REST tests are focused on the REST services, while the Selenium test examines the correctness of the operations of GUI which in turn may use the aforementioned services. The difference between these two types of tests is also noticeable in in their maintenance because of time needed to activate them. The Selenium tests are started periodically at a specified time by the continuous integration system (or manually by a developer in their environment), which prolongs the response time to errors in tests. The REST tests are executed in a continuous integration system after each commit and IN A local environment when developers build the

application, so tests errors are usually spotted immediately.

Team effort has been measured based on the time which has been logged on specific tasks which regard the creation of automated tests and their subsequent maintenance. Figure 4 presents the measured percentage of time spent on creation and maintenance related to the total time of the REST tests. A relatively low percentage of the REST tests maintenance (18%) is due to the fact that fixing is usually easy to perform. So the overwhelming amount of time is spent on the implementation of new test cases. Figure 5 presents the percentage of time spent on the creation and maintenance of the Selenium tests. Figure 6 shows the average time spent on the implementation and maintenance task of the Selenium and REST test. The average time spent by a developer on Selenium test creation (13.46h) was more than two times longer than the creation of a REST test (5.53h). An even greater difference between average times was observed between the maintenance of the Selenium and a REST tests. The average time of Selenium test maintenance tasks (6.75h) was more than three times longer than the average time logged on REST tests maintenance. It results from the fact that repairing Selenium tests is usually difficult to perform (in most cases there is a need to fix the code on both, the client and the server sides). Figure 6 shows the differences between the effort committed for the REST and Selenium tests. This is largely due to the difference in the complexity of these tests. On the other hand, the Selenium tests (which generally require more effort) detect errors in both, the server and the client side code.

In the case of the creation of both the Selenium and the REST tests, it is possible to present further statistics, see Table 1. The average numbers of hours committed to automation of a test case are reported once more but also the variabilities and results of analysing the differences between the Selenium and the REST tests are given. In the case of the REST tests not only the mean effort but also the variance is noticeably smaller. In order to compare the two types of tests a two-sample t-test was used to verify the following null hypothesis:

The average effort committed to creation of a Selenium test is equal to the average effort of a REST test creation.

versus an alternative hypothesis:

The average effort committed to creation of a Selenium test is larger than in the case of a REST test.

Table 1. Efforts committed to the creation of the Selenium and REST tests statistics

	Selenium tests	REST Tests
Mean	13.46	5.83
Variance	146.89	5.67
F	24	
$P(F \leq f)$ one-tail	0.0004	
t-Stat	3.2	
$P(T \leq t)$ one-tail	0.0014	

Since the alternative hypothesis is asymmetric, the one-tail version of the t-test was used. The hypotheses are tested at the significance level $\alpha = 0.05$. The F-Test was used to test whether the variances of two populations are equal and according to the obtained value of $P(F \leq f)$, i.e. smaller that 0.05, the two-sample assuming unequal variances version of t-test [2] was used. $P(T \leq t)$ value equal to 0.0014 was obtained, thus the null hypothesis can be rejected and the alternative one accepted. In other words, the effort needed to create a REST test is significantly smaller than it is in the case of the Selenium tests.

It is not possible to present analogous statistics for the efforts related to the maintenance of the tests. The maintenance task almost always spans across multiple test cases, hence there are no data regarding individual tests and the average values have already been reported.

The execution of the automated test cases is done in the continuous integration system, hence it does not involve additional efforts. The project compilation and build process, which is frequently executed by developers as a part of their work, includes only unit tests and REST tests and in

[2] Satterthwaite's approximate t-test, a method in the Behrens–Welch family.

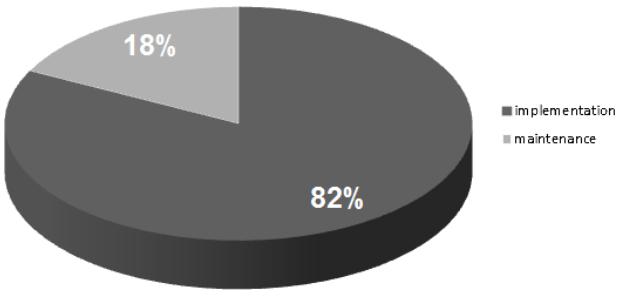


Figure 4. Time spent on the REST tests

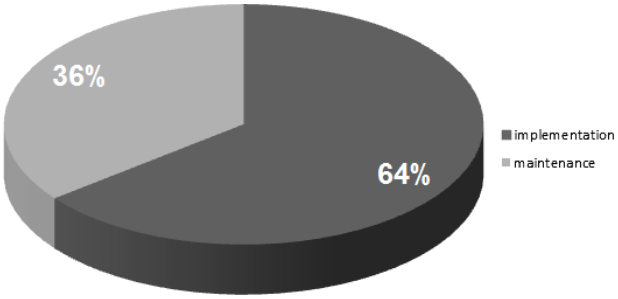


Figure 5. Time spent on the Selenium tests

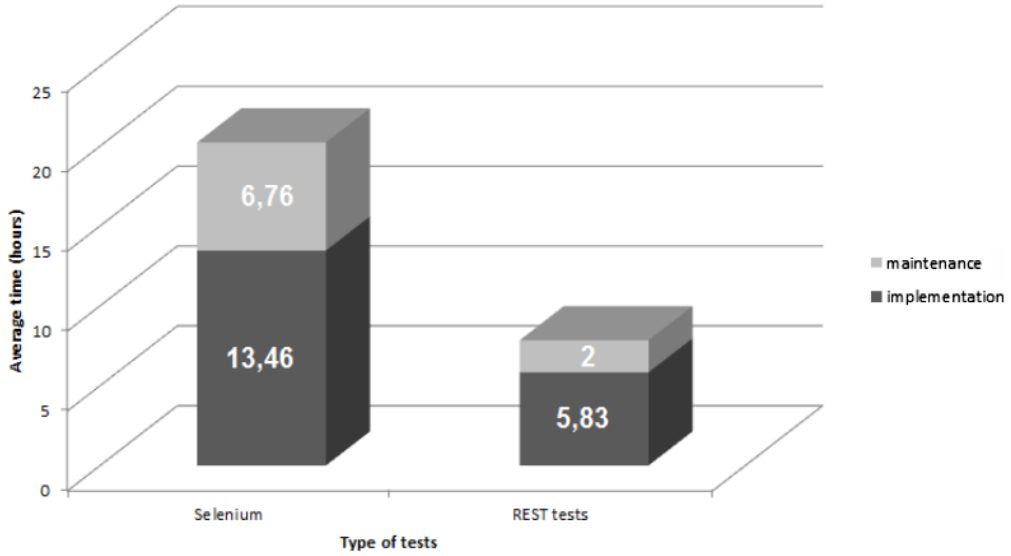


Figure 6. Average time spent on implementation and maintenance of a single Selenium and REST test

consequence it is below the 10 minutes suggested by Beck [3]. Therefore, the Authors do not consider test execution as a source of additional effort for a development team.

The next logical step in answering this research question leads to the unit tests. Unfortunately, there are no empirical data in this area. As in the case of test driven development, the unit tests are created simultaneously with the code. There were no dedicated tasks regarding unit tests that carry effort related information. Furthermore, it is not possible to decide which part of the tracked effort was committed to the production and which to the test code. However, test driven development considers the preparation of a unit test as an integral part of the development process, hence decisions regarding unit tests should not be driven by cost and effort related criteria in an agile testing process.

There is one more type of tests in the investigated project, i.e. the manual tests. These tests are outside of the scope of RQ3 as they are not appropriate for supporting unscheduled releases, but still could be interesting. Unfortunately, it was not possible to correlate the manual tests efforts per particular requirement and according to the development team there is a significant fraction of untracked efforts. Therefore, the Authors decided to report the subjective interpretations of team members involved in manual testing. The interviewed team members estimated the cost of implementing and maintaining an automated test scenario in the investigated project to be up to 20 times higher than the cost of manual execution. Furthermore, the overall testing effort (it covers both manual and automated tests) was estimated by the members of the development

team to be close to 25% of the overall development effort.

4. Threats to Validity

In this section the most trustworthy results are evaluated: to what extent they are true and not biased toward the Authors' subjective point of view.

4.1. Construct Validity

Most of the employed measures are indirect. The level of agility was assessed using questionnaires, hence there is a risk of misunderstanding the questions or giving biased answers due to unintentional company influence or context. To address the risk a meeting was organized, where all concepts from the questionnaires were explained and all questions that were answered. The possibility of unintentional influence is connected with the fact that all of the questioned people (as well as the Authors) were involved in the project development. Therefore, they have a wide knowledge about the object of study, but simultaneously they cannot have an objective point of view which is necessary to spot the influence. In consequence, it must be stated that the level of agility was assessed only from a subjective perspective. Moreover, the Authors' involvement creates an even greater threat to validity in terms of bias. To mitigate the issue quantitative measures were preferred over qualitative ones in the study design as the latter ones are more vulnerable to influence.

The ability of making an unscheduled release was assessed from the perspective of automated tests and the continuous integration system. The tests results carry information about the quality of the developed system. Nevertheless, using them exclusively is a simplification of the subject. There could be some unmeasurable factors involved (like team members intuition). The test results obtained from the continuous integration system is the most convincing, quantitative measure we were able to come up with.

Test automation efforts were evaluated using data stored in the issues tracking system. There are no serious doubts about the quality of the data, but when it comes to completeness the situation changes. There is a considerable probability that a fraction of efforts was not tracked. There could be small issues that were not reported at all or issues that were missed by the developers, e.g. if they forgot about tracking their efforts. We did not find a way to evaluate which fraction of the collected data suffers from such problems. However, the data filtering was done manually. The developers, who were assigned to the test automation tasks were approached and asked how the collected data correspond with their real efforts.

4.2. Internal Validity

According to Runeson and Höst [9] the internal validity is a concern when casual relations are examined. This study does not provide such a relation explicitly. Nonetheless, it may create an impression that following the principles that are used in the project described here should lead to similar results, which the Authors in fact believe is true. However, there is still a possibility that some relevant factors are missing as they were not identified by the Authors. To mitigate the risk of this threat, the context was described according to guidelines suggested by Petersen and Wohlin [11] and the rest of the study was designed following Runeson and Höst [9] recommendations for a descriptive and to some extend exploratory case study. Specifically, the Authors used the suggested research process, terminology, research instruments and validity analysis.

4.3. External Validity

The context of this case study is described in Section 2.1 and there is no evidence that the same or similar results could be achieved in a another context. Nonetheless, the Authors believe that the environment complexity (e.g. the need for simulators) increases the efforts related to test automation and hence better results may be obtained when there are fewer couplings with external systems.

Specifically, it must be stressed out that the comparison of efforts related to different test frameworks has very limited external validity. Statistical tests were employed, but the investigated data were mined from only one project. Therefore, it is not clear whether the results are true for other projects and it cannot be empirically verified which project specific factors are relevant for the comparison results. A plausible explanation is presented in the 'Discussion and conclusions' section, however, data from additional projects are required to confirm it.

4.4. Reliability

According to Runeson and Höst [9] reliability is concerned with the extent to which the data and the analysis are dependent on specific researchers. The conducted analyses are rather straightforward and several perspectives have been presented to ensure the reliability. Nonetheless, the Authors do not publish raw data, as they contain business critical information, and that can be an obstacle in replicating the study. Additionally, the Authors were involved in project development and thus the observations as well as conclusions may be biased – the issue is discussed in Subsection 4.1.

5. Related Work

The agile methods have been used for a couple of years. Thus, a number of case studies with regard to the testing process have already been conducted. Nonetheless, the Authors are not familiar with any works that analyze the agile testing process with respect to unscheduled releases. On the other hand, the complexity of the developed system is always a factor taken into account, however, it but seldom becomes the object of a study – none of the works reported in this section consider a similar approach to handling system complexity.

Kettunen et al. [2] compared testing processes between software organizations that applied agile practices and employ traditional plan driven-methods. Altogether twelve organizations were investigated and as a result the Authors concluded that agile practices:

- tend to allow more time for testing activates, while the total time for the project remains the same,
- smooth the load of test resources,
- require stakeholders to understand and conform to the practices in agile methods,
- are usually supported by internal customers,
- allow faster reaction time for change.

The findings advocate agile testing but do not correspond with the goals of the process investigated in this study, i.e. support for unscheduled releases and complex infrastructure.

Jureczko [5] investigated the level of agility in a testing process in a large scale financial software project. The studied project is of different size and comes from another domain, nevertheless, the suggested criteria for agility evaluation have been used to assess the testing process described in this study. Therefore, a comparison is possible. The process from [5] is significantly outperformed in the scope of continuous integration, pair programming, root-cause analysis and working software (over comprehensive documentation), however, it is underperformed in the field of communication. The overall assessment is higher in the process studied in this work. Let us elaborate the development and testing process of this work. The project was planned for 150 working years, but later the duration time was lengthened. More than 150 high skilled developers, testers and managers were involved. The project was divided into five sub-projects and the paper is focused on only one of them. The sub-project was developed by a group of about 40 people. The project is a custom-build solution that supports more than 1500 initially identified case scenarios. The system is based on a well defined technical framework, called Quasar. The development team frequently delivers new releases with new functionalities. There are two major releases per year: in spring and autumn. They are used to deliver large changes and bulks of possible bugs. The two major ones are supplemented by hotfix releases that target the most important bugfixes only. The employed testing process forces

practices borrowed from the V-Model. Testers work on test concepts once the specification of a requirement is ready. Subsequently developers write a source code and perform manual tests when testers write new automated tests. Each of the tests is immediately added to the regression test set that is executed daily. Subsystem tests are performed after the integration. They are usually manual and focused on new features.

The role of test automation in a testing process was empirically evaluated by Karhu et al. [7] in five software organizations among which he was developing a complex system with a number of interfaces to customer-specific external systems and hence creates challenges in test automation. The Authors identified a number of interesting consequences of automation. Quality improvement through better test coverage and increase in the number of executed test cases were noted among benefits, whereas costs of implementation, maintenance and training were pointed out as the main disadvantages. Moreover, according to Berlino [1] great emphasis is put on this issue and the methods of extending the degree of attainable automation are in the focus of testing research. Unfortunately, there is a dichotomy with regard to the industrial reality, i.e. many companies, which claim that they have adopted XP, practice the automation in a limited scope [19]. Considerable research has been conducted on the test automation [16, 20–23] and in general this practice is strongly recommended. Among the aforementioned works especially [21] is noteworthy as it is conducted it the context of the Scrum method. Another commonly investigated agile testing practice is test driven development. There is evidence for its usefulness in the industrial environment [24, 25], reports of controlled experiments are also available [26, 27].

Winter [28] investigated the challenges regarding testing in a complex environment, however, the complexity came from evaluation usability for a variety of end users. One of the areas of interest was the agility of a testing process and the balance between the formal and informal approaches. The agility was considered in the context of the Agile Manifesto, and hence there was limited overlap with the criteria employed in our study (i.e. only Working software (over comprehensive documentation) and Responding to change (over following the plan) are considered in both studies).

Talby et al. [6] described installation of agile software testing practices in a real large-scale project in a traditional environment. The authors pointed out that in the investigated case study the agile methods dramatically improved quality and productivity. The testing process was analyzed and described in four key areas: test design and activity execution, working with professional testers, planning, and defect management. The investigated project is a sub-project of a larger one. The larger one was developed by 60 skilled developers and testers. Thus, there is a considerable probability that the sub-project is similar in size to the project described in this study. More details about the software project investigated in [6] can be found in [29].

A lesson learned from a transformation from a plan-driven to agile testing process is documented in [14]. Extreme programming and Scrum were adopted, which makes the process similar to the one described here. Hence, the challenges described by Puleio [14] may arise when trying to adopt adopt the principles advocated in this paper in a traditional (i.e. not agile) environment.

6. Discussion and Conclusions

This paper contributes to the body of knowledge in two ways. The Authors provide a detailed case study of an agile testing process, which can be used in further research and in combination with other studies to help make general conclusions. The documentation of the testing process in this project could also be beneficial to practitioners interested in installing an agile testing process in a similar environment. Especially, a number of ready to use testing tools are recommended (e.g. the simulators).

A survey was conducted in order to assess the level of agility and the results showed a high level of adoption, i.e. 76.6%. Furthermore, each of the assessment criteria was compared against the

project reality in a qualitative approach which gives an insight into the way the agile principles work. A detailed analysis indicated some areas of possible improvement, e.g. communication with customer and pair programming.

The project preparation for the 'on short notice' release was analysed and assessed according to test execution results in the continuous integration system. The results gave a value of about 47% of time in which the project was ready to be released and 10% of time it could have been released with the acceptable risk. Hence, the Authors can conclude that the requirement for unscheduled releases is supported to a considerable extent.

Comparative evaluation of the cost of the REST and Selenium tests was conducted. It measured how much time is necessary in both test frameworks for the implementation of new test and maintenance of the existing ones. A significant disadvantage was found in the case of the Selenium tests, which we believe is a result of using Google Web Toolkit for the graphical interface – this framework uses dynamic identifiers and generates complex (at least from the Selenium point of view) web pages. This extra cost is counterbalanced with an ability to detect bugs in GUI which is covered only by the Selenium test. Nonetheless, the big difference in costs encourages reconsideration of the testing approach.

A brief description of all tools that were used to implement automated tests is provided. The main contribution in this area regards the suggestion for mitigating environment complexity with simulators. There is a detailed description of an open-source project called SSH-simulator which was co-developed by the Authors, in fact, it is officially presented in this paper for the first time. Furthermore, the Authors also suggested a straightforward solution for simulating WebServices in a test environment. The paper contains listings that show how to mock a WebService in the context of the JUnit tests. The Authors believe that the detailed descriptions of those simulators will help other practitioners who face similar challenges during test automation since the presented solutions are ready to use (the necessary listings and references to external sources are given in Sec. 3.2.5).

The overall results are satisfactory with regard to the project goals. Therefore, we would like to recommend following the same rules (in similar projects), i.e. adopt the principles of agility, specifically assure high quality and coverage of automated tests and employ a continuous integration system for automated builds.

References

[1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering, FOSE '07*. IEEE, 2007, pp. 85–103.

[2] V. Kettunen, J. Kasurinen, O. Taipale, and K. Smolander, "A study on agility and testing processes in software organizations," in *Proceedings of the 19th international symposium on Software testing and analysis*. ACM, 2010, pp. 231–240.

[3] K. Beck and C. Andres, *Extreme programming explained: embrace change*. Addison–Wesley Professional, 2004.

[4] L. Koskela, *Test driven: practical TDD and acceptance TDD for Java developers*. Manning Publications Co., 2007.

[5] M. Jureczko, "The level of agility in the testing process in a large scale financial software project," in *Software engineering techniques in progress*, T. Hruška, L. Madeyski, and M. Ochodek, Eds. Oficyna Wydawnicza Politechniki Wrocławskiej, 2008, pp. 139–152.

[6] D. Talby, A. Keren, O. Hazzan, and Y. Dubinsky, "Agile software testing in a large-scale project," *IEEE Software*, Vol. 23, No. 4, 2006, pp. 30–37.

[7] K. Karhu, T. Repo, O. Taipale, and K. Smolander, "Empirical observations on software testing automation," in *International Conference on Software Testing Verification and Validation, ICST '09*. IEEE, 2009, pp. 201–209.

[8] K. Schwaber, *Agile project management with Scrum*. Microsoft Press, 2004.

[9] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, Vol. 14, No. 2, 2009, pp. 131–164.

[10] R. Mall, *Fundamentals of software engineering*. PHI Learning Pvt. Ltd., 2009.

[11] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proceedings of the 3rd International Symposium on Em-*

pirical Software Engineering and Measurement. IEEE Computer Society, 2009, pp. 401–404.

[12] S. Harichandan, N. Panda, and A.A. Acharya, "Scrum testing with backlog management in agile development environment," *International Journal of Computer Science and Engineering*, Vol. 2, No. 3, 2014.

[13] K.K. Jogu and K.N. Reddy, "Moving towards agile testing strategies," *CVR Journal of Science & Technology*, Vol. 5, 2013.

[14] M. Puleio, "How not to do agile testing," in *Agile Conference*. IEEE, 2006, pp. 305–314.

[15] K. Beck, *Test driven development: By example*. Addison–Wesley Professional, 2003.

[16] M. Jureczko and M. Mlynarski, "Automated acceptance testing tools for web applications using test-driven development," *Electrical Review*, Vol. 86, No. 09, 2010, pp. 198–202.

[17] S. Ambler, *Agile modeling: effective practices for extreme programming and the unified process*. Wiley, 2002.

[18] C. Kaner, J. Bach, and B. Pettichord, *Lessons learned in software testing*. John Wiley & Sons, 2008.

[19] D. Martin, J. Rooksby, M. Rouncefield, and I. Sommerville, "'Good' organisational reasons for 'bad' software testing: An ethnographic study of testing in a small software company," in *29th International Conference on Software Engineering, ICSE 2007*. IEEE, 2007, pp. 602–611.

[20] M. Catelani, L. Ciani, V.L. Scarano, and A. Bacioccola, "Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use," *Computer Standards & Interfaces*, Vol. 33, No. 2, 2011, pp. 152–158.

[21] R. Löffler, B. Güldali, and S. Geisen, "Towards model-based acceptance testing for Scrum," *Softwaretechnik-Trends*, Vol. 30, No. 3, 2010.

[22] X. Wang and P. Xu, "Build an auto testing framework based on selenium and fitnesse," in *International Conference on Information Technology and Computer Science, ITCS 2009*, Vol. 2. IEEE, 2009, pp. 436–439.

[23] T. Xie, "Improving effectiveness of automated software testing in the absence of specifications," in *22nd IEEE International Conf. on Software Maintenance, ICSM '06*. IEEE, 2006, pp. 355–359.

[24] N. Nagappan, E.M. Maximilien, T. Bhat, and L. Williams, "Realizing quality improvement through test driven development: results and experiences of four industrial teams," *Empirical Software Engineering*, Vol. 13, No. 3, 2008, pp. 289–302.

[25] A.P. Ress, R. de Oliveira Moraes, and M.S. Salerno, "Test-driven development as an innovation value chain," *Journal of technology management & innovation*, Vol. 8, 2013, p. 10.

[26] L. Madeyski, "The impact of pair programming and test-driven development on package dependencies in object-oriented design—an experiment," in *Product-Focused Software Process Improvement*. Springer, 2006, pp. 278–289.

[27] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Information and Software Technology*, Vol. 52, No. 2, 2010, pp. 169–184.

[28] J. Winter, K. Rönkkö, M. Ahlberg, and J. Hotchkiss, "Meeting organisational needs and quality assurance through balancing agile and formal usability testing results," in *Software Engineering Techniques*. Springer, 2011, pp. 275–289.

[29] Y. Dubinsky, D. Talby, O. Hazzan, and A. Keren, "Agile metrics at the Israeli air force," in *Agile Conference, Proceedings*. IEEE, 2005, pp. 12–19.