Systematic Analysis of Practical Issues in Test Automation for Communication based Systems

Nikolay Tcholtchev, Martin A. Schneider, Ina Schieferdecker
Fraunhofer Institute for Open Communication Systems FOKUS, Berlin, Germany
{nikolay.tcholtchev, martin.schneider, ina.schieferdecker}@fokus.fraunhofer.de

***Abstract*— This paper is about issues experienced along testing large-scale industrial products, with safety and security critical relevance. The challenges in testing - several thousand requirements for several product variants and various configurations - were addressed by test execution automation. However, since principal testing concepts as well as architectural concepts were ignored or poorly implemented, the test automation activities faced various difficulties within the considered projects. The current paper presents these issues in an abstracted manner and discusses possible solutions.**

***Keywords* — industrial experiences, test automation, large scale industrial projects, telecommunication infrastructures, railway domain**

I. Introduction & Motivation

In various domains, test execution automation (in short test automation in the following) has significantly gained on importance as the means to ensure the quality of IT and communication based products. This includes telecom and Internet equipment, embedded systems as well as any sort of application domain, where digital communication and handling of digital data plays a significant role.

There are various approaches to test automation including the usage of legacy programming languages (e.g. Java, Python, Perl, C/C++, …) and corresponding frameworks such as JUnit [2] and CUnit [3]. Furthermore, such test suites are often deeply embedded within the development process and automatically executed during code development. This is in particular facilitated by continuous integration [5] platforms such as Hudson [4] or Jenkins [1].

In addition to the above-described aspects, various standardization and certification bodies provide test suites and test means, which are meant to help vendors and are further used to certify a product in a particular scope. Examples for such bodies are given by the IPv6 Forum [6] and ETSI [7]. The IPv6 Forum maintains its IPv6 Ready Logo [8] program, which allows certifying communication equipment - being it a router, host, embedded or mobile device - with regard to its IPv6 readiness. ETSI is considered with standard test suites for a large variety of domains and protocols, such as automotive, mobile communication (such as 3G or 4G), Internet protocols (such as IPv6 or SIP), All-IP-convergence including the IP Multimedia Subsystem, M2M (Machine-2-Machine) communication, etc.. Since this type of test suites is not necessarily deeply embedded within the development processes and are having more of an acceptance test character[1], other more abstract languages such as TTCN-3 [9] come into the spotlight. TTCN-3 stands for *Testing and Test Control Notation version 3*, and is a test definition language that is strongly pushed by ETSI, thereby undergoing a tremendous further development with respect to research and standardization activities in the past years. TTCN-3 is used for a variety of standardized test suites across the world, including the examples of the IPv6 Forum and ETSI with its relevant sub-groups and activities. In the case of the IPv6 Forum, certified testing labs in cooperation with the IPv6 Forum, provide both TTCN-3 based solutions [11] as well as test solutions [10] using legacy programming languages such as Perl.

In addition, the topic of Model-based-Testing (MBT) is increasingly receiving attention by industrial partners, with various research prototypes and test solutions in place [12][13]. For example, the UML Testing Profile (UTP) [14], which is developed at the OMG (Object Management Group), provides a promising framework for the generic development of test architectures and test cases/scenarios with the final goal of test code generation (e.g. TTCN-3 or JUnit), execution and reporting. The UTP approach can be further combined with aspects such as risk analysis, thereby improving the test automation and enabling efficient test management, with respect to various constraints such as security and safety critical aspects [15].

In the current paper, we present - in anonymized form - our practical experiences from two large-scale test automation projects and propose measures on how to avoid some of the main problems that hindered the related activities. Thereby, the problems are identified based on selected test automation patterns [16], which are discussed and developed within the test automation community.

[1] Even though such test suites and test means are often also used by the Quality Assurance departments in companies.

The rest of this paper is organized as follows: Section II presents briefly the two reference projects, from which the experiences were gathered. Section III presents the evaluation results for each project based on selected test automation patterns from the relevant community. Section IV provides some lessons learnt on how to avoid the experienced problems. The final section summarizes our experiences and draws conclusions.

II. Reference Projects

The two reference projects took place in parallel for a period of one and two years, respectively. The descriptions are abstracted and anonymized so as to help extracting the lessons learnt without giving unnecessary details on the involved companies and products.

Both projects used test automation as a necessary means for checking the quality of the system under test (SUT), which was the target of evaluation/development, and different interesting issues were experienced. The first project relates to a component that was developed within the railway domain. The second project is given by component tests within a telecommunications infrastructure for the eHealth application domain.

A. Component Tests in the Railway Domain

The SUT from the railway domain – as pictured in its test environment in Figure 1 – is a component running on a real-time operating system. Basically, it serves as a gateway that establishes the communication between a train control and management system (TCMS) controlling different elements of a train, such as doors, drive, brakes, and human-machine-interface, and a railway control system, that enables safe operation when recognizing dangerous situations, usually by stopping the train using an "emergency brake". Thus, the SUT constitutes a safety-critical component on the safety integrity level 2, where 1 is the lowest level and 4 is the highest, leading to specific requirements for instance on signal processing and transmission, and more generally on the whole development process of such a component.

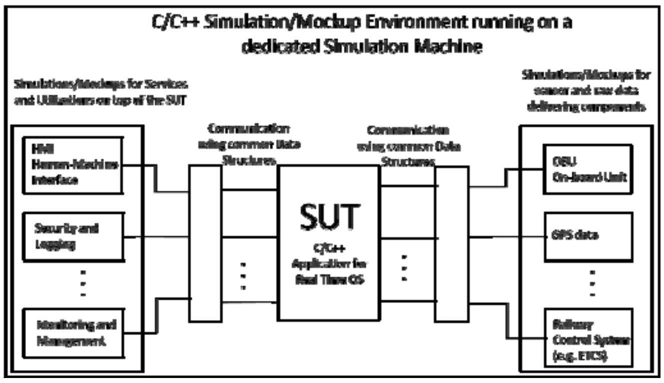
C/C++ Simulation/Mockup Environment running on a dedicated Simulation Machine
Simulations/Mockups for Services and Utilizations on top of the SUT
Simulations/Mockups for sensor and user data delivering components
HMI Human-Machine Interface
Security and Logging
Monitoring and Management
Communication using common Data Structures
Communication using common Data Structures
SUT
C/C++ Application for Real Time OS
OBU On-board Unit
GPS data
Railway Control System (e.g. ETCS)

FIGURE 1: TEST AND SIMULATION ARCHITECTURE FOR THE COMPONENT TEST WITHIN THE RAILWAY DOMAIN

Since the SUT is performing a safety-critical task, thorough testing is required to ensure that no faults lowering the safety level remain, and to achieve the regulatory requirements. The SUT was developed in several releases where new functionality was stepwise added. In addition to testing the newly developed functionalities, regression testing was critical to detect side-effects of new code to existing parts of the SUT and retests to ensure that revealed faults were actually fixed. Due to the high number of requirements, test cases, signals, and messages, a test automation solution was inevitable.

B. Component Tests in the eHealth Domain

The SUT in the second case study is constituted by a hardware component, which is required to connect the front end of an integrated eHealth solution with its backend applications and services (including network services). Thereby, the goal of the component is to act as a secure and trusted proxy that protects the communication between front-end and back-end on several layers of the ISO/OSI stack, ranging from network and link layer aspects up to the services and applications layer. This includes the usage of VPN tunnels, TLS communication for REST calls, trusted DNS and DNSSEC, trusted NTP communication, certificate management and handling, as well as secure routing to external networks (e.g. Internet or special data networks).

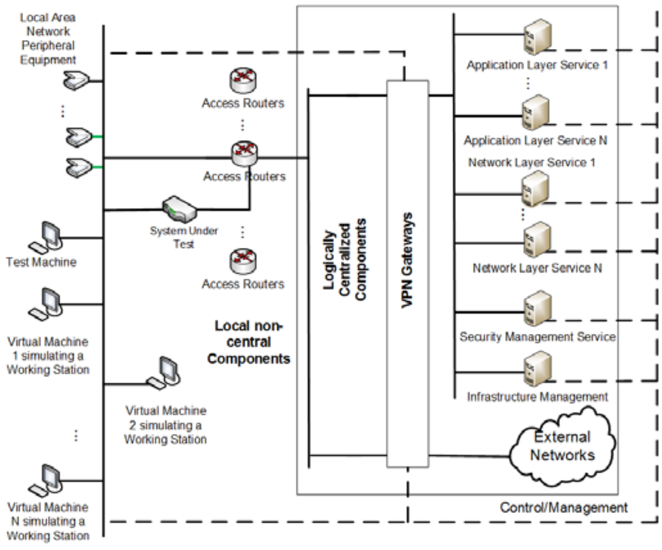
Local Area Network Peripheral Equipment
Access Routers
Access Routers
System Under Test
Access Routers
Test Machine
Virtual Machine 1 simulating a Working Station
Local non-central Components
Virtual Machine 2 simulating a Working Station
Virtual Machine N simulating a Working Station
Logically Centralized Components
VPN Gateways
Application Layer Service 1
Application Layer Service N
Network Layer Service 1
Network Layer Service N
Security Management Service
Infrastructure Management
External Networks
Control/Management

FIGURE 2: TEST AND SIMULATION ARCHITECTURE FOR THE COMPONENT TEST WITHIN THE LARGE SCALE TELECOMMUNICATIONS INFRASTRUCTURE

The test architecture for this component test is roughly drafted in Figure 2. It consists of a large number of virtual machines, which are used to enable various scenarios and aim at simulating real world components. The used test architecture comes with a very high complexity of roughly 20 test machines, 30-35 test components (this means software modules running on the VMs), several additional forwarding proxies (HTTP, SOCKv4, ..) and around eight network segments. On the left side, one can observe the simulators (VMs) for work-stations and peripheral equipment in LAN network segments, whilst on the right side we observe the telecommunications infrastructure (without going into details) for the secure communication to the backend – including access routers, VPN gateway, network and application level services. These services are realized by open source components (e.g. StrongSwan [17] for the VPN aspects), which on one hand has saved quite some effort in implementing mock-ups, but on the other hand imported all the problems of these open source solutions (e.g. non-conformance to standards for some aspects).

III. Evaluation Results

The current section focuses on systematically understanding the experiences from the two reference projects by applying the test automation patterns, which are continuously developed and collected by the community on the belonging *Test Automation Patterns* portal [16]. The test automation patterns are classified into patterns for test processes, for the management thereof, for the test automation design, and for the execution of the tests.

In the course of the analysis, we applied the definitions from [16] on our experiences and obtained a list of issues, which indicated the points with potential for further improvement. Subsequently, the key issues were picked and are presented in the following sub-sections. This systematically demonstrates how an industrial project can be viewed through this prism of Test Automation Patterns [16] with the goal of identifying potential improvements.

The overall result of this analysis is constituted by the identification of key problems with respect to how test automation is utilized in industry at the current point in time. These identified aspects are further utilized for the purpose of improved quality assurance for communication-based systems.

TABLE 1: TEST AUTOMATION ISSUES OF THE COMPONENT TEST WITHIN THE RAILWAY DOMAIN

No	Issue	Manifestation	Impact/Effect	Frequency
1	INADEQUATE COMMUNICATION (Testers don't know what automation could deliver and the test automation team doesn't know what testers need or Developers don't understand, don't know or don't care about the effect of their changes on the automation)	Interfaces of the system under test are changed by name and/or type to a large extent without considering and communicating the effects on test automation	Much effort in order to update the test environment and test cases with nearly no or little benefit	Regularly
2	NO INFO ON CHANGES (Development changes are not communicated to the test automators, or not in good time.)	Developers changes semantic of values without updating the interface specifications or informing about this change in any other way	Test cases fail due to outdated test cases leading to bug reports that has to be analyzed by developers	Regularly
3	BRITTLE SCRIPTS (Automation scripts have to be reworked for any small change to the Software Under Test (SUT))	Test cases contain complex sequences with many details to be checked in order to test the interaction of the system under test with an HMI	Each change of requirements leads to updates of a large number of complex test cases leading to significant delays.	Seldom
4	FALSE FAIL (The tests fail not because of errors in the SUT, but because of errors in the test automation testware or environment issues)	Test cases pass when run in the whole test suite but fail when run separately	Inability to decide whether the tested feature was correctly implemented or not	Regularly
5	INADEQUATE RESOURCES	Test execution is done on old devices and using inadequate equipment leading to significant delays when stimulating the system under test on the target platform	Test execution on the target hardware was impossible, and thus, the available test automation solution was not usable	Constantly
6	KNOW-HOW LEAKAGE (Test automation know-how (for instance scripting, tools) is being lost from the organization.)	Test automation know-how is lost due to personnel leaves the team and no documentation of the test automation solution was created	Updates of the test environment requiring additional effort if new software versions and configurations have to be tested, moreover, missing documentation led to approval issues	Constantly

A. Applying the Test Automation Patterns to the Project from the Railway Domain

Although a dedicated test automation solution, specifically adapted to testing the components of the TCMS was available, this solution was not applicable to the described SUT because it differs with respect to the tools used for its implementation. Although significant effort was invested, no viable solution was achieved to circumvent this issue by executing the SUT on the target platform. Real-time requirements on input data from the railway control system and insufficient computational power of the test devices to cope with the real-time requirements lead to issue no. 5 (see Table 1). This prevented the test execution with the SUT on the target platform and, thus, with the "close-to-real-world" application of the existing test automation solution.

The whole SUT was executed on a traditional PC. The developed test solution, depicted in Figure 1, establishes the communication between the SUT and the test environment via shared common data structures used for stimulating the SUT with certain input data and observing the SUT's reaction for test verdict arbitration. Several mock-ups replaced relevant components of the TCMS and the railway control system that communicate via the SUT responsible for pre-processing of data and transforming it to the required data format of the communication partner (refer to Figure 1).

Furthermore, this dedicated test automation solution lead to issue no. 6 (see Table 1): A single test engineer developed this solution. Due to tight development schedule, lack of awareness of this solution considering it as an interim solution, there was no documentation created. Know-how leakage was the result as the responsible person left the team, leading to additional work, in order to adapt the test automation solution to new interfaces, delaying releases of new versions and variants of the SUT due to missing knowledge on how and where to perform the corresponding changes.

As described above, the SUT has many signals (inputs as well as outputs) to be stimulated and observed during test execution. The development team lacks awareness of effects of their changes to the test team. This lead to issues no. 1 and no. 2 where changes to the interfaces of the SUT were not or only insufficiently communicated, resulting in additional effort spent to adapt the interfaces of the test automation solution to the changes of SUT and to investigate why previously passing test cases now fail without functional changes of the SUT, respectively.

Due to complex interaction with the SUT required for reaching a certain state, in particular with respect to the human-machine-interface, a lot of test scripts were implemented by copy and pasting test code that brings the SUT to such a state resulting in issue no. 3. Each time a change of the SUT has led to a different communication sequence to reach the required state, a lot of test scripts had to be adapted, in order to reflect these changes, leading to substantial delays with an increased risk of incorrect adaptions that, in turn, led to an increased effort for investigating the reason why test scripts are failing.

Last but not least, each time a regression test was performed for a new release of the SUT, all test cases were executed in sequence, where each test case resets the test environment and the SUT. In addition, each test case was executed on its own by stopping the test automation solution completely after a test case has been executed. It was observed that test results differed for some test cases when executed separately compared to the execution within the sequence of test cases. This led to false fails (issue no. 4) due to misunderstood interactions of the SUT with its environment, which resulted in inadequate reset mechanisms of the test environment. In the first place, it was therefore impossible to decide whether a tested feature was correctly implemented or not.

B. Applying the Test Automation Patterns to the eHealth Project

Based on the systematic view implied by [16], the seven main potential points for improvement that were experienced within the eHealth project are presented in Table 2. The main problem was given by the extremely complex test environment (see the drafted testbed in Figure 2). As a result, the focus shifted from managing the SUT to managing and fixing problems in the testbed. Such problems were given by routing problems, connectivity problems, IP addressing issues, crashed software (e.g. Web servers) and inconsistent states of the involved test machines/counterparts (excluding the SUT) in general. The complex environment – analyzed under *COMPLEX ENVIRONMENT* in Table 2 – has led to the regular occurrence of further issues such as *ERRATIC TEST*, *FALSE FAIL* and *FALSE PASS*. The complexity of the testbed and the resulting inconsistencies combined with the problems on judging whether a failed test is due to the SUT or due to the testbed issues, has brought severe problems in the course of failure analysis w.r.t. to bug fixing. This aspect is described in the *INEFFICIENT FAILURE ANALYSIS* row in Table 2 (issue no. 4). Together with the *NO INFO ON CHANGES* (issue no. 7), the abovementioned aspects resulted in the fact that the test automation was not really usable for regression testing of new product versions/releases. The *NO INFO ON CHANGES* aspect reflects on the fact that the test automation made use of an internal API (for the SUT), where changes were very dynamic and were not communicated to other stakeholders within the product eco-system. Since this API was an internal one, changes to it were meant to be a point of interest only for the involved core developers, but not for the testers, which led to serious issues whilst trying to test a new component version/release with the test automation.

The fact that regression testing was very difficult and even impossible to conduct resulted in a disappointment on project management and stakeholder level, with respect to project controlling and tracking of key performance indicators. The high expectations, which were projected on the test automation, were not met. Correspondingly, the management was unwilling to invest additional resources for fixing the main aspects which led to the fact that regression testing became even more difficult by a high degree of manual testing. These aspects are reflected in the HIGH ROI EXPECTATIONS pattern (issue no. 6 in Table 2).

Based on the considerations in this section, the following paragraphs summarize the findings in the form of proposed solutions with the goal to improve the observed test automation practices.

TABLE 2: FOR THE COMPONENT TEST WITHIN THE LARGE SCALE TELECOMMUNICATIONS INFRASTRUCTURE

No	**Issue**	**Manifestation**	**Impact/Effect**	**Frequency**
1-3	ERRATIC TEST (The automated tests seem to pass or fail randomly) FALSE FAIL (The tests fail not because of errors in the SUT, but because of errors in the test automation testware or environment issues) FALSE PASS (The tests pass even if the SUT actually reacts erroneously)	The first attempted test automation was having the property that test cases were randomly going to pass or fail. It is suspected that this was due to: - too complex test environment and inconsistent state of roughly 20 test machines, 30-35 test components, several forwarding proxies (HTTP, SOCKv4, ..) and around eight network segments. - strong reliance on timers and timing/performance aspects of the product even in situations where it is possible to correctly obtain whether a particular process was accomplished or not	The test automation was not usable for the purpose of regression testing and analysis of the product quality for new versions.	Regularly (for the duration of the project)
4	INEFFICIENT FAILURE ANALYSIS (Failure Analysis is slow and difficult)	The complex nature of the test environment - 20 test machines, 30-35 test components, several forwarding proxies (HTTP, SOCKv4, ..) and around eight network segments – has made it extremely difficult to identify whether a failures has its root cause in the SUT or in the test system as a whole.	The test automation solution could not be used for regression testing of new product versions. Only limited usage for some specific tests was achieved.	Regularly (for the duration of the project)
5	COMPLEX ENVIRONMENT (The environment where the Software Under Test (SUT) has to run is complex)	- the automation is based on too complex test environment a.k.a testbed, which is clearly overdimensioned for module/component test and even goes strongly into the direction of interoperability test - the complex testbed leads to difficulties in diagnosing whether the cause for a failed test case is within the SUT or the testbed (test environment), due to said complexity - the attempted test automation embeds complex pre-coditions which constitute a test themselves. These complex test conditions, if failed, block automatically a lot of test cases and do not allow any agile testing and regression in the course of the development process - usage and development of complex testing components instead of simple mockups that just return the required messages for each test case	The test results were not trusted at all, since it was impossible to identify whether a test case failure is due to an SUT problem or a test environment problem.	Regularly (for the duration of the project)
6	HIGH ROI EXPECTATIONS (Management wants to maximize ROI from the test automation project, but is not prepared to invest adequately)	The management layer was expecting some high level KPIs to be met, with less reflection on the complexity and time resources required for the test automation.	Test automation failed partially because of the high return of investment expectations (from management's side) without the readiness to invest enough time and resources.	Regularly (for the duration of the project)
7	NO INFO ON CHANGES (Development changes are not communicated to the test automators, or not in good time.)	Severe issues due to the usage of a proprietary non-external (for the SUT component), internal API which was constantly changing There was a constant problem that changes in the API, which was used for the management of the component/module were not	This turned to be a major problem that implicated that the test automation could not be used after new versions of the SUT were delivered. Correspondingly, the test automation could not be utilized for regression testing.	Regularly (for the duration of the project)

		communicated to the test automators. Changes did not include only the functions or messages for the API, but also the behavior of the functions (e.g. synchronous vs asynchronous). On the other hand, this API is not a standard (e.g. IETF standard as SNMP) and was never contractually agreed to be provided to externals. This management API was meant only for internal use within the different processes inside the component being developed (i.e. inside the SUT).	This issue also increased the mistrust in the verdict results (FAIL or PASS) of the test automation output.	

IV. Lessons Learnt

The current section summarizes a list of lessons learnt and possible solutions by applying resolving patterns for the issues identified in the previous sections. The identified aspects are related to one of the reference case studies and aim at providing a remedy for the identified problems.

In the railway case study, many of the test automation problems resulted from mainly two issues: inadequate resources (issue no. 5 in Table 1) and inadequate communication (issue no. 1).

Solution 1: Resolving Pattern 'Dedicated Resources'
By providing information and resources (personnel as well as technical) in an early stage to the test automation team, a viable test automation solution could be developed for the SUT.

On the other hand:

Solution 2: Resolving Pattern 'Whole Team Approach'
Test automation awareness across the teams and thus, resulting in better communication between developers and testers would significantly reduce efforts for maintaining the test automation solution and test scripts and thus, would enable to meet the release schedule.

With respect to the eHealth project, the aspects and solutions from above are also relevant and valid. In addition, the complexity of the test environment turned out to be a critical aspect that hindered many of the subsequent activities and desirable results.

Solution 3: Resolving Pattern 'Keep It Simple'
The test environment for component tests should be kept as simple as possible, in order to avoid the focus shifting from the management and controlling of the SUT to the management and controlling of the testbed.

Furthermore, given the usage of virtualization technology, snapshots for virtual machines can be used in order to enable the efficient restoring of test environment states, and thus being able to efficiently pin-point whether a test failure is due to the test environment or the SUT.

Solution 4: Resolving Pattern 'Fresh Setup'
In case of complex testbeds, increase the usage of virtualization technology and snapshots of virtual machines and networks. That way the configurational states and the complexity of the test environment and testbed will be better managed/handled, since the expected testbed configuration will be automatically in place (using snapshots) before each test case run.

In addition, within a component test, the usage of open source software as a testing means/counterpart for some of the interfaces turned out to be cumbersome given the problems, which were imported by using this software. Furthermore, the SUT adapts to the problems and specifics of the used open source counterpart, which already contains some aspects and issues related to interoperability (testing) and goes beyond the pure component conformance test. Hence, the usage of mockups should be encouraged, when performing conformance tests on component/module level.

Solution 5: Resolving Pattern 'Do a Pilot'
Increase the usage of simple easily manageable request-response mockups in the scope of component tests.

Finally yet importantly, it should be concluded that there should be overall readiness for an increased investment in the initial phase of a test automation development. This increased investment pays back later when it comes to conducting regression testing and continuously assessing the quality of the components in question. These considerations lead to further *potential solutions*:

Solution 6: Resolving Pattern 'Share Information'
Increase the readiness and adapt the controlling and management KPIs for a larger initial investment in test automation and track the ROI along regression testing.

Solution 7: Resolving Pattern 'Automate Good Tests'
The usage of regression testing for the continuous quality evaluation of the SUT should be a fundamental strategic goal of the Quality Assurance team. This goal is achieved by a careful consideration and implementation of a practical and manageable test automation solution.

V. Conclusions

The current paper presented a systematic analysis of practical issues related to test automation, which were experienced in two large-scale industrial projects. The projects are anonymously described, in order to give the reader a high-level understanding regarding the level of complexity, as well as the safety and security criticality of the components under test. Subsequently, a systematic analysis of the faced issues was conducted based on a pattern catalogue of test automation practices, which is pulled together and maintained within the test automation community. In large, the two case studies not only showed a clear ignorance of a substantial set of best practices (as reflected by the test automation patterns), but also confirm the failure patterns thereof on *Going for the Numbers* and *Tool Mushrooming*.

Finally, several possible solutions are proposed. The main point remains to be the increase of awareness and communication between development and test automation team, as well as the need to keep the testbed environment simple and easily manageable, in order to efficiently handle the testing processes – including continuous regression testing. This is as such not a revolutionary new insight, but demonstrates the need to educate and train development and test teams in good test automation practices and cooperative work. Parts of this are addressed by the upcoming ISTQB [18] advanced level syllabus on test automation.

References

[1] Jenkins Continuous Integration Server: http://jenkins-ci.org/, as of date 04.02.2016

[2] JUnit Testing Framework: http://junit.org/, as of date 04.02.2016

[3] CUnit testing Framework: http://cunit.sourceforge.net/, as of date 04.02.2016

[4] Hudson CI: http://hudson-ci.org/, as of date 04.02.2016

[5] Fowler, Martin, “Practices”, “Continuous Integration”, online: http://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration, as of date 04.02.2016

[6] IPv6 Forum: http://www.ipv6forum.com/, as of date 04.02.2016

[7] ETSI (European Telecommunications Standards Institute): http://www.etsi.org/, as of date 04.02.2016

[8] IPv6 Ready Logo Program: https://www.ipv6ready.org/, as of date 04.02.2016

[9] TTCN-3 ETSI page: http://www.ttcn-3.org/. as of date 04.02.2016

[10] TAHI-TestSuite: http://www.tahi.org/logo/phase2-core/, as of date 04.02.2016

[11] IPv6 Ready Logo TTCN-3 Test Suite: https://www.irisa.fr/tipi/wiki/doku.php/conformance_and_interoperability_testing, as of date 04.02.2016

[12] Marc-Florian Wendland, Jürgen Großmann, Andreas Hoffmann: Establishing a Service-Oriented Tool Chain for the Development of Domain-Independent MBT Scenarios. ECBS 2010: 329-334

[13] Ina Schieferdecker, Axel Rennoch, Alain Vouffo-Feudjio: Model-Based Testing: Approaches and Notations. Encyclopedia of Software Engineering 2010: 571-576

[14] UML Testing Profile: http://www.omg.org/spec/UTP/, as of date 04.02.2016

[15] Shaukat Ali, Tao Yue, Andreas Hoffmann, Marc-Florian Wendland, Alessandra Bagnato, Etienne Brosse, Markus Schacher, Zhen Ru Dai: “How Does the UML Testing Profile Support Risk-Based Testing”, ISSRE Workshops 2014: 311-316

[16] Test Automation Patterns: https://testautomationpatterns.wikispaces.com/, as of date 04.02.2016

[17] StrongsWan: https://www.strongswan.org/, as of date 04.02.2016

[18] International Software Testing Qualifications Board: http://www.istqb.org, as of date 03.03.2016