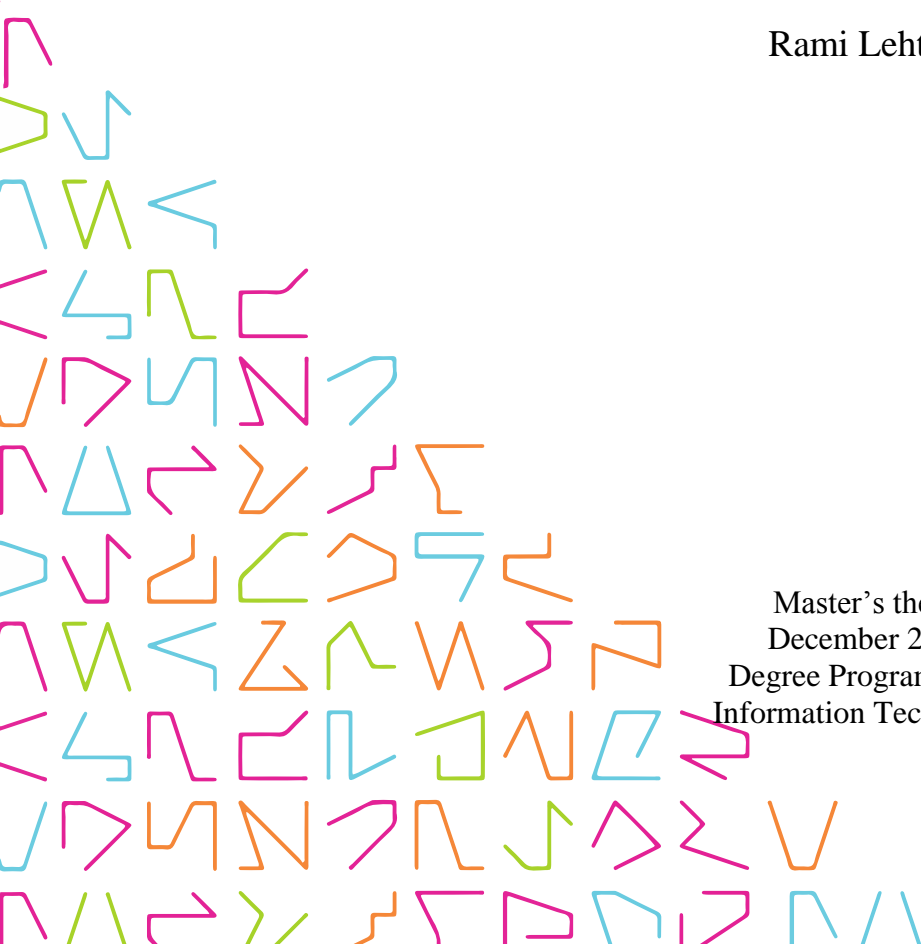


TEST AUTOMATION SOLUTION FOR WEB APPLICATIONS IN WIN- DOWS ENVIRONMENT

Rami Lehtelä

Master's thesis
December 2017
Degree Programme in
Information Technology



ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Information Technology

LEHTELÄ, RAMI:

Test Automation Solution for Web Applications in Windows Environment

Master's thesis 47 pages, appendices 11 pages
December 2017

There has been a need to verify functionality of web application in an end-user like environment. Until now all automated tests are run against web applications in Linux environment on Mozilla Firefox web browser. This setup doesn't match the majority of end-user environments in the field.

The purpose of this project was to create solution into continuous integration pipe where web applications can be tested automatically in web browsers running on Windows operating system. Requirement was to create solution which can be easily taken into use from one web application testing to another. Focus was defining Windows environment and applications needed to enable test execution in Windows environment. Also, connectivity and file transfer issues needed to be solved between Windows and Linux systems.

First Windows environment was specified so that it would have possibility to run Robot Framework test automation cases and allow remote connection from Jenkins running on Linux. Image of the specified Windows was then introduced to vCenter where it could be cloned to virtual machines by request and accommodate parallel test execution. The test execution orchestration was implemented into Jenkins, which was responsible of requesting clone of the Windows environment, communication and transferring test files to and from Windows environment, and visualization of test results.

As a result, the test automation solution was introduced into continuous integration pipe as a separate entity. Test automation possibility for web browsers running on Windows environment have already caught some faults specific to Windows web browsers, which could not have been detected in Linux environment. Test results were visualized in Jenkins, where developers and verification engineers can monitor the quality of the product.

It is crucial to automate as much as feasible when testing software. This project will act as an enabler when moving from continuous integration towards continuous delivery, where web applications must be verified in end-user like environment. It was noted that automating test execution and maintaining Windows environment is really complicated and time consuming. Although the maintenance load of this solution seems big, it was a must to develop. In the future, there might be alternative solutions to automate web application testing in Windows environment using containers. Containers have become more and more popular in software development and should be considered as a candidate also in Windows environment.

Key words: web application, test automation, window environment, continuous integration, continuous delivery

CONTENTS

1	INTRODUCTION	7
2	TEST PIPE COMPONENT REQUIREMENTS	9
2.1	High-level requirement description	9
2.2	Used technologies	10
2.2.1	Revision control system	10
2.2.2	Computing platform	10
2.2.3	Test automation framework	11
2.2.4	Automation server	11
2.3	Windows test environment	11
2.3.1	Technical details of Windows test execution environment requirements	12
2.4	Jenkins configuration and test result visualization	12
2.4.1	Jenkins job step configuration.....	13
2.4.2	Test result visualization.....	13
2.5	Scripting requirements	14
2.6	Test suite directory requirements.....	15
3	TEST PIPE IMPLEMENTATION	17
3.1	Windows environment setup	17
3.1.1	Windows Server 2008 R2 installation.....	17
3.1.2	Windows configuration.....	18
3.1.3	Setup for cloning purposes.....	23
3.2	Jenkins job definition and implementation.....	23
3.2.1	Jenkins job configuration	23
3.2.2	Implementation and functions of the mpp.properties file	27
3.3	Test suite directory implementation.....	27
3.3.1	Suite structure.....	27
3.3.2	Files	28
3.4	Test run launch possibilities	29
4	TEST EXECUTION AGAINST EXAMPLE APPLICATION	30
4.1	Description of example application	30
4.2	Content of the example test suite directory	31
4.2.1	Starting example test run on Windows environment.....	32
4.3	Test result and visualization	32
4.3.1	Test result view on Jenkins job page	32
4.3.2	Test report and log html-files.....	33

5	FEEDBACK AND ANALYTICAL COMPARISON WITH PREVIOUS TEST SETUP	36
5.1	Benefits of new solution	36
5.1.1	Test execution environment	36
5.1.2	Test execution orchestration	37
5.1.3	Test suite directory	37
5.2	Challenges in the new solution	38
5.2.1	Test execution environment	38
5.2.2	Test execution orchestration	39
5.3	End user feedback about test pipe.....	40
5.3.1	Feedback from SW architect.....	40
5.3.2	Feedback from SW engineer.....	40
5.4	Project outcome compared to high-level requirement description	41
5.5	Reasoning.....	42
6	FUTURE USE CASES	43
6.1	Next steps to get full potential out of the Windows environment	43
6.2	Windows environment test automation on different verification steps ...	43
6.2.1	Daily regression testing step	44
6.2.2	QL4 testing step	44
6.3	Front end development possibilities	45
6.4	Other applications outside of current development area.....	45
6.5	Macintosh operating system	46
6.6	Windows web browser testing and docker containers	46
	REFERENCES.....	47
	APPENDICES	48
	Appendix 1. TestRunnenr.java	48
	Appendix 2. pom.xml	49
	Appendix 3. mpp.properties	51
	Appendix 4. tc_ifa_supercell_legacy.tsv test case file.....	54
	Appendix 5. download_copy_webdrivers.sh	55
	Appendix 6. robot_run.bat.....	56
	Appendix 7. Feedback from SW architect	57
	Appendix 8. Feedback from SW engineer	58

ABBREVIATIONS AND TERMS

API	Application Programming Interface
clone	In the context of this thesis clone is copy of a virtual machine configuration.
Cygwin	Tool that provides Linux like functionalities on Windows
ISO	Refers to ISO image, binary image of an optical media file system (usually ISO 9660 and its extensions of UDF)
HW	Hardware
Java	Computer programming language
Jenkins	Open source automation server which can be used to compile and test SW source code
JRE	Java Runtime Environment
Linux	Unix-like computer operating system
LTE	Long Term Evolution, is a standard for high-speed wireless communication for mobile devices and data terminals.
macOS	Macintosh operating system developed by Apple Inc.
MPP	Merlin Production Pipe, Nokia internal project for collection of integration tools
OpenSSH	Is the premier connectivity tool for remote login with the SSH protocol.
OS	Operating System, for example Linux, Windows and macOS
PC	Personal Computer
Pip	package management system used to install and manage software packages written in Python
Python	Computer programming language
QL4	Quality Level 4, Internal quality level which indicates the quality of system component. Is also used as a quality gate for system component promotion.
RAM	Random Access Memory
Robot Framework	Keyword driven generic test automation framework for acceptance testing
SCP	Is a network protocol which supports file transfer between hosts on a network.

SSH	Secure Shell, cryptographic network protocol for operating network services securely over an unsecured network
SUT	System Under Test, system that is being tested for correct operation. In this case application under test
SVN	Apache Subversion, software versioning and revision control system
SW	Software
tar	is a computer software utility for collecting files into a one archive file usually .tgz
tarball	
TSV, .tsv, tsv	Tab-separated values file is a simple text format used for test suites run in Robot Framework
UDF	Universal Disk Format
UI	User Interface
Unix	Multi-tasking multiuser computer operating system
WebDriverManager	Open source program used to update and download latest web browser drivers

1 INTRODUCTION

The purpose of this project was to create a solution how to automate test execution of web applications running on web browsers in Window operating system. By taking Windows environment test automation in use, the testing is done in an operating system closest to end-user like environment. This solution will serve as a quality assurance step when moving to continuous delivery mode of software development.

The project will go through the steps from configuration of test execution environment to test result visualization. This report will introduce the used technologies which will determine the way, how the development this test automation solution will be implemented. One of the main objectives in this work is to create end-user like Windows environment on which the web application could be tested. Other main topics are solving the communication issues between Linux and Windows environments and how to run tests on Windows environment remotely. The test automation solution is proven by using Google Chrome as a web browser in which web application is used by test automation framework. The report contains parts that are applicable for Firefox and Internet Explorer, but are not documented to avoid duplicity. The result is test automation solution that can be launched from Jenkins and is easily modifiable to accommodate web application testing requirements.

This thesis is written in consideration of software professionals having knowledge about continuous integration and test automation. This report will give guideline what to take into consideration when implementing a test automaton solution for Windows environment. This thesis can be used as a step by step guide to recreate similar solution or software professional can take parts of the thesis, which can be used in some other solution. Report contains some example code and scripts which could be modified and taken into use for example when automating Google Chrome update procedure.

The thesis has a chapter where the test automation solution is proved by testing already existing web application and test results are shown. Challenges and benefits of this solution were analyzed and it was found that there is reasoning for using this solution even

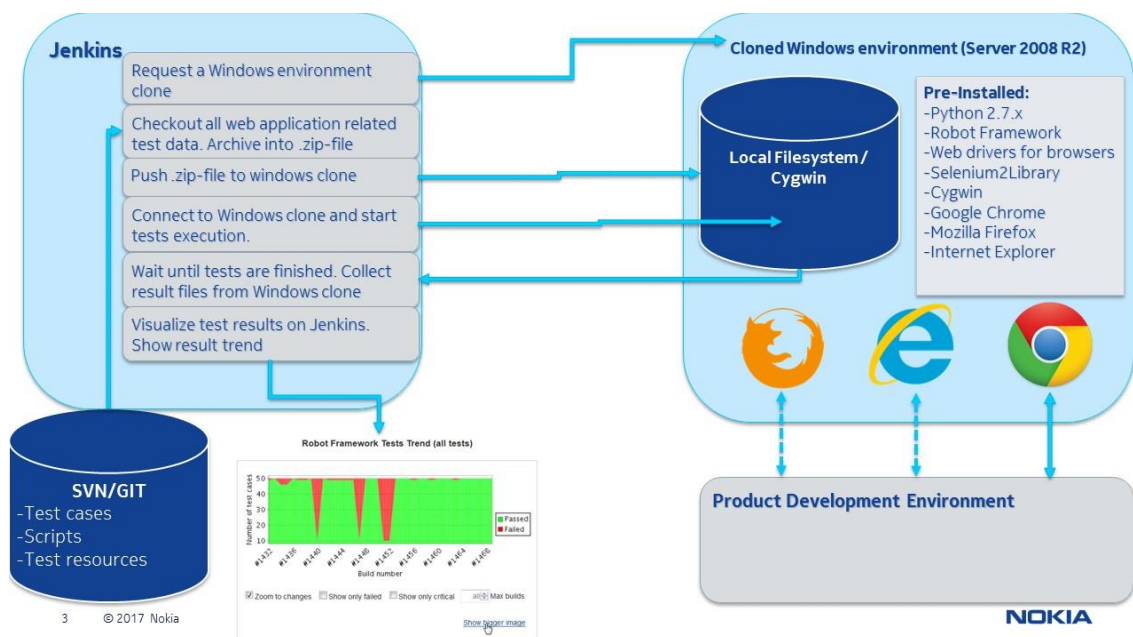
though maintenance load using Windows is relatively big. This project considers the future aspects containing front-end development usage, test automation solution usage in regression testing phase, and implementation of web browser testing using containers.

2 TEST PIPE COMPONENT REQUIREMENTS

This chapter will go through requirements and used technology solutions for test automation and continuous integration pipe components. The components will contain revision control system, automation server, test execution environment, test result visualization, SUT (System Under Test), and interaction between all entities.

2.1 High-level requirement description

High-level requirement description is made to give a clear image what are the implementation steps to create test automation solution for web widgets in Windows environment. In the picture 1, the illustration of a complete test automation solution is shown. Requirement contains main building blocks of the solution like: used revision control system, automation server and needed functions, Windows environment and needed applications, and communication and file transfer needs on each step.



PICTURE 1. High-level illustration of test automation solution requirement

Requirements for the entities shown in picture 1 are explained in more detail in the following subheadings in this chapter.

2.2 Used technologies

In a large company, there are many technologies, tools and solutions that have been selected as main components to be used in production. Usually freeware solutions are most desired as they are free of charge and can be easily configured to serve specific use cases needed.

2.2.1 Revision control system

Apache Subversion (SVN) is selected for the revision control system for our production and is used also for this project. SVN is an open-source, centralized version control system provided by Apache software foundation. For this project SVN is easy to take into a use since all used technologies and tools have SVN plugin in our system and test cases and resources can be checked out with already used commands. (Apache Subversion, 2017)

2.2.2 Computing platform

VMware vSphere provides virtualization which is an abstraction layer that breaks the hard connection between physical HW (Hardware) and OS (Operating System). Virtual machine is a SW (Software) computer that is like a normal physical computer running OS but has virtual HW allocated from the vCenter server. Huge number of virtual machines can easily be deployed and cloned depending on the need. (VMware vSphere, 2017)

Windows OS will be installed on virtual machine in cloud computing virtualized platform vSphere provided by VMware. vSphere client is already used in our production to host our product running on distributed virtual machines. vSphere client is also used to clone and take snapshots from Nokia network management system product

2.2.3 Test automation framework

Robot Framework is a generic test automation framework for acceptance testing and can be used web application end-to-end testing. Robot Framework is keyword driven framework which can be extended with external libraries containing keywords for specific types of testing (Robot Framework, 2017)

2.2.4 Automation server

Jenkins is the leading open source automation server, which can be used for building and to automate any project (Jenkins, 2017). Jenkins is selected to orchestrate all SW related functions from compiling the source code to run end-to-end test automation against SUT. Test result visualization will follow the basic principles that Robot Framework will provide.

2.3 Windows test environment

Since Windows is the most commonly used OS for PC-clients, there is a need to select the OS versions that are in use now when planning test execution environment. Windows server clients are easier to setup and configure for a company having huge amount of already existing Windows server licenses than the more traditional PC Windows OS's. For each PC Windows client, there is an equivalent Windows server OS release that is more feasible to be used. Windows 7 is the most commonly used Windows OS in our customer base currently, but latest Windows OS releases must be taken into consideration. Windows client PC releases and corresponding Windows server releases are mapped in Table 1.

TABLE 1. Corresponding Windows client and server releases

Windows PC client release	Windows server release
Windows 7	Windows Server 2008 R2
Windows 8	Windows Server 2012
Windows 10	Windows Server 2016

The approach chosen is to start setting up Windows Server 2008 R2 release to be able to verify product in a Windows 7 like environment. All other Windows environment configurations can be configured using VMware vSphere client when functionality is proven with Windows Server 2008 R2 release.

2.3.1 Technical details of Windows test execution environment requirements

The Windows test execution environment – later in the text “Windows environment” – must be reachable from the networks where SVN and Jenkins are located. Windows parent configuration environments will be configured using the VMware vSphere client. VMware tooling enables Windows environment parent to be easily cloned when Jenkins is requesting a Windows environment. This will support high number of parallel test executions, since vSphere can host many Windows environment simultaneously. Windows environments must have a way to connect with Jenkins using SSH, since Jenkins is running on Linux. From the resources point of view Windows environment should have enough memory assigned for the graphic card to support minimum resolution of 1280x1024 pixels. Other resource requirements like processor and RAM (Random Access Memory) amount must meet typical Windows 7 PC configuration to ensure as close as possible end-user like environment for the test case execution.

Windows environment should have Python, Robot Framework including latest browser drivers and needed keyword libraries, latest versions of main web browsers (Google Chrome, Internet Explorer and Firefox), and Java JRE (Java Runtime Environment). These requirements will provide an end-user like system on which tests can be run using Robot framework against main web browsers. Web browsers should also be updated frequently so that the test automation cases can identify bugs if they arise after browser update.

2.4 Jenkins configuration and test result visualization

To orchestrate the test execution and test result visualization Jenkins will be used. Jenkins should be configured so that there is a job dedicated to run tests against SUT. In this case web widget or web application.

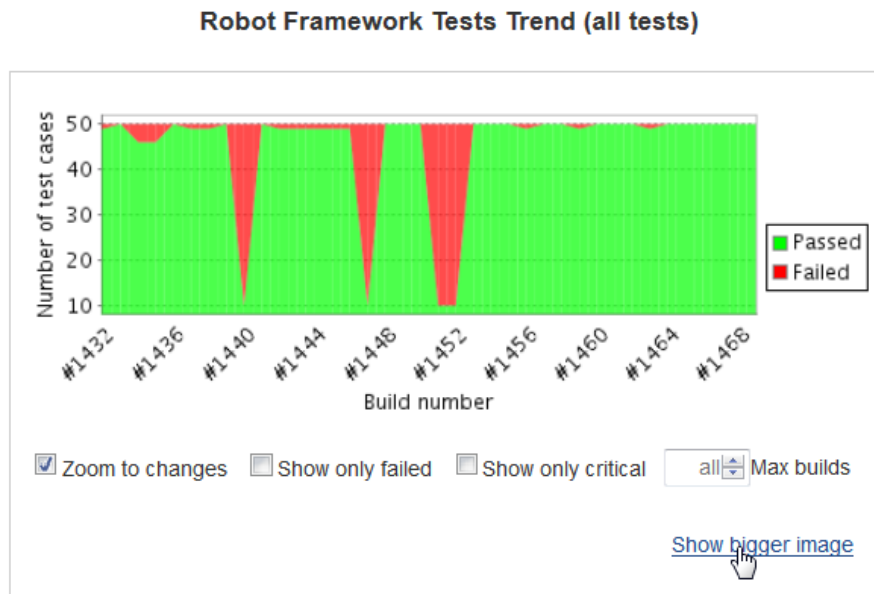
2.4.1 Jenkins job step configuration

There will be certain steps that must be implemented into Jenkins job. First Jenkins job must be set up with SVN locations from where all the needed data can be checked out into Jenkins workspace. From SVN test suite directory containing test cases, test data, resources, and scripts will to be checked out. Test suite directory is later pushed to the Windows environment. The other entity that is needed from SVN will contain test tool scripts which will enable communication and file transfer between Jenkins and Windows test execution environment over the SSH protocol. Another SVN location will store mpp.properties file which will be used as a main script to perform the test execution orchestration between Jenkins and Windows test execution environment.

Communication and file transfer will be defined by mpp.properties file which will allow easy way to configure the Jenkins job without touching Jenkins job configuration at all. Last configuration step is to add Robot Framework plugin to the build main page to enable result visualization and linkage for the test report files.


2.4.2 Test result visualization

Jenkins job must have Robot Framework plugin configured to enable the test result visualization. Robot Framework plugin provides nice legend of test cases run pass-fail-ratio trend and provides links to results of the test run. Robot Framework test trend from the Jenkins job main page can be seen in picture 2.



PICTURE 2. Robot Framework Tests Trend.

In addition of the Robot Framework test trend, the result reports and logs must be saved into Jenkins workspace to provide the means to backtrack possible failed test cases. Robot Framework plugin should provide the view that can be found from picture 3.



Latest Robot Results:

	Total	Failed	Passed	Pass %
Critical tests	48	13	35	72.9
All tests	49	13	36	73.4

- [Browse results](#)
- [Open report.html](#)
- [Open log.html](#)

PICTURE 3. Latest Robot Results.

2.5 Scripting requirements

Scripting will be needed to automate certain functions in the Jenkins. The `mpp.properties` file contains set of functions which will be used to communicate with Windows environment. These functions will contain file transfer, remote execution, Windows environment clone request and release scripts. Some of these scripts are ready made internally and will

be used in addition of newly created scripts. This collection of scripts in mpp.properties will be run on every time Jenkins job is started. Jenkins offers many different build steps, but was decided to use shell scripting. Because all scripting files and libraries are in revision control and can be easily modified without changing Jenkins job configuration.

Scripts will be used in the Windows environment to execute certain functions and to prepare the web browser. Web browser in Windows environment require web driver that enables test execution with Robot Framework. Web browser and web driver are loosely tied together, but if either one of those gets out of compatibility between each other tests can't be executed. To overcome this issue web driver and web browser must be updated before test execution. This is done by creating a script that downloads the web driver and starts browser specific update executable. Web driver executables must be added into path. When Robot Framework is started and Selenium2Library is used to start web browser, Robot Framework will look path if the needed web driver is present. To overcome this issue a copy step should be implemented into script run in Windows environment. Last step in the script run in Windows environment will be the start of Robot Framework test execution.

2.6 Test suite directory requirements

In addition of test suites containing test cases the directory should contain all needed resources, test data and scripts which are necessary for test case execution. Test suites contain testcases which are collection of keywords. These keywords are defined in keyword libraries or can be resource files which consists of combined keywords for specific tasks. All the resource files containing keywords used in test cases must be in the same directory. Keywords from the libraries that can be downloaded using pip into Robot Framework don't need to be in the test suite directory, but can be imported into test suite using only the library name. Downloaded libraries will be added into Python27 folder which is usually defined in environmental variable "Path".

Every test suite directory must have test data if the functionality of the target web application is tested. Test data should be in own directory in the same directory structure. To enable test with different amount of data, it reasonable to have different test data sets in

the directory. Test data should be imported into SUT with readymade keywords that can be found from resource files.

Test suite directory will also need the scripts to run tests, download web drivers and updating the web browsers. These scripts must be compatible in the Cygwin and in Windows environment itself. This might give challenges to find most reasonable way to execute all the needed steps to run tests in Windows environment. Scripts should contain shell and batch scripts. All the scripts must have execute-rights. Without execute rights there won't be possibility to launch test execution remotely from Jenkins.

3 TEST PIPE IMPLEMENTATION

This chapter goes through all implementation steps from scratch to ready-made end-to-end test pipe solution. Implementation was divided into individual entities and will be taken into a closer look. The main issue of the project was enabling and configuring the ways to communicate efficiently between Linux and Windows based systems. Test result visualization and possibility to introduce this test automation solution using Windows environment to other teams outside our own organization was one of the principles.

3.1 Windows environment setup

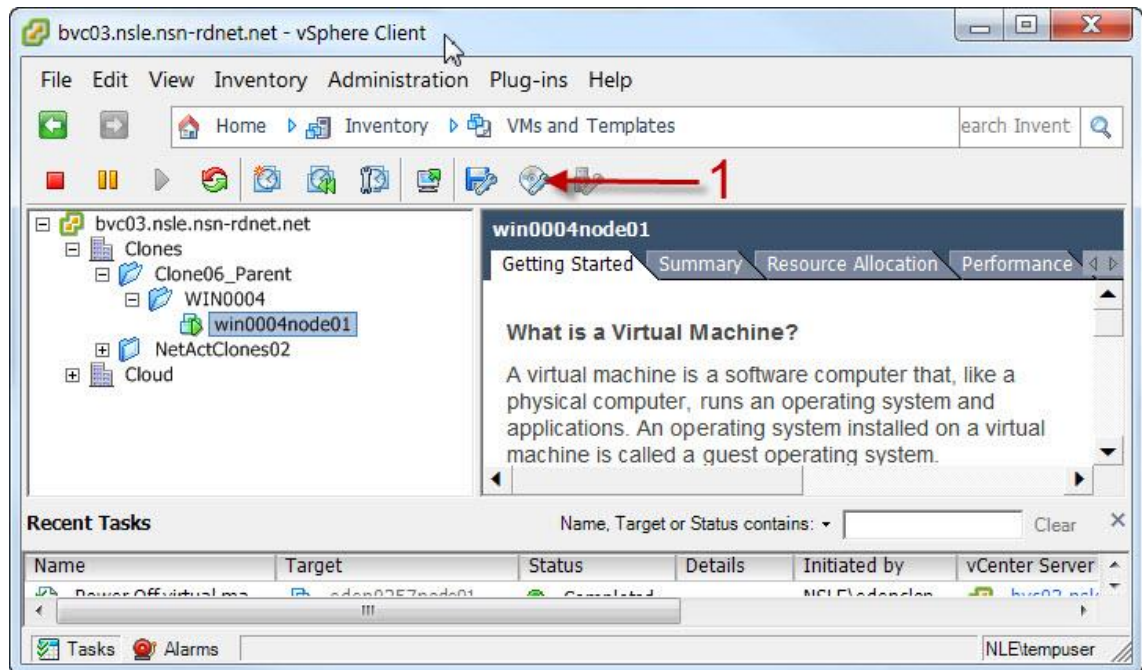
This chapter will dive into the Windows environment setup taking closer look at SW and applications that needed to be installed on top of Windows environment. Installed SW was a collection of end user programs used for web browsing and test automation tools. Windows Server 2008 R2 installation on VMware virtual machine will be gone through first in brief.

3.1.1 Windows Server 2008 R2 installation

vSphere Client was used to create a new virtual machine in which the parent Windows environment can be installed. First vSphere client was connected to the vCenter server where virtual machine can be created. The connected vCenter server address can be seen on title bar of the vSphere Client in picture 4. After the virtual machine was created and can be seen in the picture 4. with name “win004node1” Windows Server 2008 R2 can be installed on it. An ISO image of parent Windows environment was needed so it can be mounted into a virtual machines CD/DVD device to start the installation. ISO image can be mounted by clicking icon pointed by number 1 in the picture 4.

After successful installation of the parent Windows Server 2008 R2 OS the virtual machine was ready to be configured. Install/Upgrade VMware Tools needed to be run from Inventory -> Virtual Machine -> Guest in vSphere Client to enable desktop view in the

Console Tab. When desktop can be accessed in the Console tab Windows Remote Desktop connection can be enabled in the installed Windows environment. This eased the configuration of the Windows test environment remotely from users own client PC.



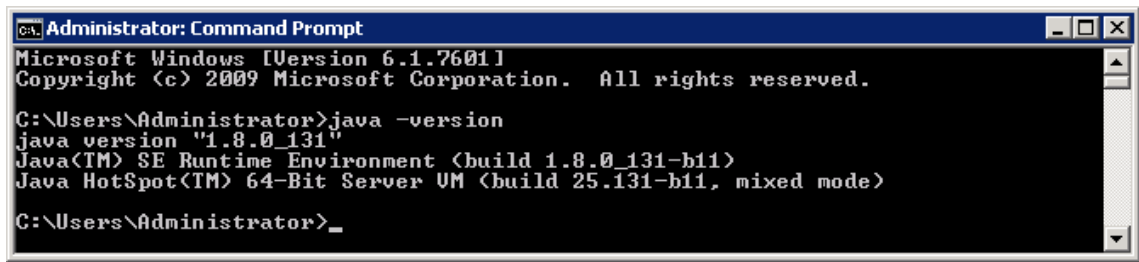
PICTURE 4. vSphere client

3.1.2 Windows configuration

To be able to execute tests in Windows environment there were certain amount of applications that provided the means to run automated test cases – in this case Robot Framework. Installation and configuration of each application and configuration is explained in detail below.

Java JRE

Latest version of JRE was installed to be able to run WebDriverManager to download latest web drivers for each web browser. Java JRE was downloaded from Oracle's Java JRE 8 download page (Java SE Runtime Environment 8 Downloads). JRE installation was executed using Windows installer with default settings. Java JRE installation was verified by running "java -version" command in Windows Command Prompt. Outcome of the command can be found from picture 5.



```

Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)

C:\Users\Administrator>_

```

PICTURE 5. Java JRE installation verification.

Robot Framework and Python

Python 2.7.12 release was installed to enable execution of Robot Framework tests. Python Windows installer was downloaded from (Downloads Python | Python.org). Since Python 2.7.12 contains pip it was used to download and install Robot Framework and all needed libraries in Windows command prompt. Following commands in Table 2 were used to install Robot Framework 3.0, Selenium2Library, SSHLibrary, and SeleniumLibrary:

TABLE 2. Pip installation commands.

Command	Action
pip install robotframework	Installs the latest Robot Framework.
pip install robotframework-selenium2library	Installs the latest Selenium2Library.
pip install robotframework-seleniumlibrary	Installs the latest SeleniumLibrary.
pip install robotframework-sshlibrary	Installs the latest SSHLibrary.

After installation of Python and Robot Framework including libraries the installation was verified by typing “python --version” and “robot -- version”. All installed libraries were listed using command “pip list”. Installed Python, Robot Framework and libraries can be seen in the picture 6.

```

Administrator: Command Prompt
C:\Users\Administrator>python --version
Python 2.7.12

C:\Users\Administrator>robot --version
Robot Framework 3.0 (Python 2.7.12 on win32)

C:\Users\Administrator>pip list
DEPRECATION: The default format will switch to columns in the future. You can use
 --format=(legacy|columns) (or define a format=(legacy|columns) in your pip.conf
 under the [list] section) to disable this warning.
cffi (1.9.1)
cryptography (1.6)
decorator (4.0.10)
enum34 (1.1.6)
idna (2.1)
ipaddress (1.0.17)
paramiko (2.0.2)
pip (9.0.1)
pyasn1 (0.1.9)
pyparser (2.17)
robotframework (3.0)
robotframework-selenium2library (1.8.0)
robotframework-seleniumlibrary (2.9.2)
robotframework-sshlibrary (2.1.2)
selenium (3.0.2)
setuptools (20.10.1)
six (1.10.0)

C:\Users\Administrator>_

```

PICTURE 6. Verification of installed Python, Robot Framework and libraries.

WebDriverManager

WebDriverManager is an open source program that provides the means to download and update web browser drivers in the beginning of each test run. To take WebDriverManager into use it was introduced as a new dependency into Maven project in specific pom.xml file. After defining the WebDriverManager dependency, TestRunner.java class needed to be added into pom.xml as well. TestRunner.java uses functions from WebDriverManager package and contains functions that enable which browser drivers will be downloaded. After all dependencies and build information were present in the maven project defined in pom.xml, “mvn package” command was run and “webdrivermanager-1.0-SNAPSHOT.jar” file can be found from target folder. This .jar-file is then used to download and update web drivers by running command “java -jar webdrivermanager-1.0-SNAPSHOT.jar <browser>”. The executable “webdrivermanager-1.0-SNAPSHOT.jar” file was committed into test suite directory SVN location so it can be used in Windows environment during test execution. TestRunner.java class can be found in appendix 1. and maven project pom.xml in appendix 2.

Cygwin

Cygwin was chosen to be installed on Windows test execution environment to enable connectivity between Windows and Jenkins running on top of Linux using SSH. Cygwin contains many GNU and open source tools by default. Some additional tools were chosen

to be installed to facilitate for example archiving the files and remote connection by configuring openSSH suite into Cygwin. Installing the OpenSSH and OpenSSL was done by starting the Cygwin's setup.exe file. In the setup phase OpenSSH and OpenSSL were selected from the package selector. When packages were installed the Cygwin was started from desktop as Administrator. After Cygwin has started the OpenSSH was configured using step-by-step instructions from (How To Get SSH Command-Line Access to Windows 7 Using Cygwin). These instructions contain: enabling the sshd run as a service, user and password creation, and starting the sshd service. Configuration was verified after the setup by running "ssh -v localhost" command. In the picture 7 the successful SSH-connection to localhost can be seen with debug messages.

```

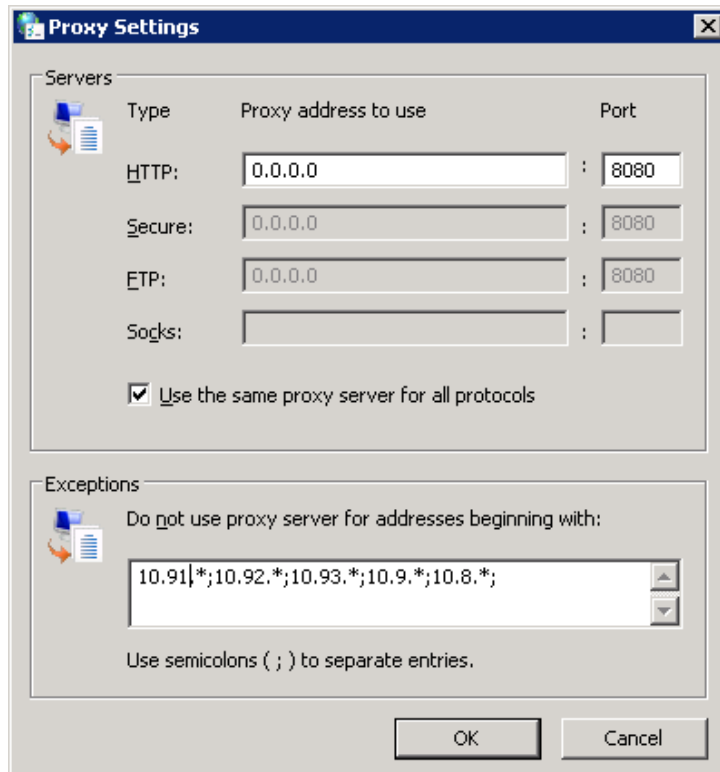
Administrator@WIN-VABN5HT3KN6 ~
$ ssh -v localhost
OpenSSH_7.5p1, OpenSSL 1.0.2k 26 Jan 2017
debug1: Reading configuration data /etc/ssh_config
debug1: Connecting to localhost [::1] port 22.
debug1: Connection established.
debug1: identity file /home/Administrator/.ssh/id_rsa type 1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/Administrator/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.5
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.5
debug1: match: OpenSSH_7.5 pat OpenSSH* compat 0x04000000
debug1: Authenticating to localhost:22 as 'Administrator'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256 SHA256:UngcQgFRbLHpL8IO7oiaIrYbFKxXR64hxfzRNYb/05U
debug1: Host 'localhost' is known and matches the ECDSA host key.
debug1: Found key in /home/Administrator/.ssh/known_hosts:1
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/Administrator/.ssh/id_rsa
debug1: Server accepts key: pkalg rsa-sha2-512 blen 279
debug1: Authentication succeeded (publickey).
Authenticated to localhost ([::1]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
Last login: Mon Jul 3 14:02:59 2017 from ::1

```

PICTURE 7. SSH-connection verification to localhost.

Network configuration

After verification of the SSH-connection to localhost, network configurations in Windows were changed so that the connection to and from outside networks is also working. Internal proxy server was configured into Local Area Network (LAN) Settings. There was also a need to add some networks into proxy settings so that proxy is not used for those. Example proxy settings can be seen in picture 8 containing HTTP proxy and exception networks in which proxy is not needed.



PICTURE 8. Example proxy settings with networks without proxy.

Web browser installations

Google Chrome, Mozilla Firefox and Microsoft Internet Explorer were installed into Windows environment to facilitate testing the functionality of the product in different web browsers. Browsers were configured so that the automatic updates were on – if possible – to keep them up to date. Some of the browsers were unable to update themselves since the test cases execution started too fast after the Windows environment clone was started. To overcome this issue a few more lines of code were needed into test execution start script. The lines of code included starting of the browser and polling the update executable from windows task list. The test case execution was only started after the update process was stopped for the browser.

3.1.3 Setup for cloning purposes

After the Windows environment was configured so that it contained all needed applications and tools it's good to make a copy of that. After copying the ready Windows environment template, snapshot was taken from the Windows environment. Windows environment that the snapshot was taken needed to be shut down when cloning was taken into use. When snapshot was available the clone could be created based on that snapshot.

The idea of having another Windows template alongside the other from which clones were made, was to be able to modify to configuration if needed and install new updates. Whenever there was a need to change something in the test execution environment this combination could be swapped and new snapshot be taken. After swapping the snapshots the new clones have updates on them and were ready to be cloned.

3.2 Jenkins job definition and implementation

To start automated end-to-end testing in Jenkins, new Jenkins job needed to be created and configured. For Jenkins job creation and configuration Jenkins Tutorial was used as a reference (Jenkins Tutorial, 2017). Following sections will go through Jenkins job configuration and logic used in scripts run by Jenkins.

3.2.1 Jenkins job configuration

After new Jenkins job was created it needed to be configured to execute series of tasks which contain: requesting a Windows environment clone, downloading test data and test cases from SVN, downloading scripts used for communication and file transfer between Jenkins and Windows environment from SVN, starting the test execution, and visualization of test results. This Jenkins testing job won't have any dependencies to other build and testing jobs. Tests run in this job can be scheduled to run when ever needed or triggered by a user.

When new job was created, name was assigned for the project and some description. In addition, two users were assigned with configure rights for this job. These settings can be seen in picture 9 with other configuration options.

The screenshot shows the Jenkins job configuration page for 'tools-widget-tester'. The page has a dark header with a search bar, the user 'Lehtelä Rami', and a 'log out' link. Below the header, the page title is 'configuration'.

The configuration fields are as follows:

- Project name:** tools-widget-tester
- Description:** This job is running tests against web widgets on top of windows. Job details: -request windows environment clone, -gather test files from svn, -trigger tests on windows, -get test results from windows, -visualize test results. There is a '[PegDown] Preview' link below the description.
- Enable project-based security:** (checked)
- Block inheritance of global authorization matrix:** (unchecked)
- Authorization Matrix Table:**

User/group	Credentials		Job							Run	Promotion	SCM					
	Create	Delete	Manage Domains	Update	View	Build	Cancel	Configure	Delete	Discover	Move	Read	Workspace	Delete	Update	Promote	Tag
rihtela	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
saollika	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
- User/group to add:**
- Block build if certain jobs are running:** (unchecked)
- Discard Old Builds:** (checked)
- Strategy:** Log Rotation (dropdown menu)
- Days to keep builds:** 30 (input field)

Below the 'Days to keep builds' field, there is a note: 'if not empty, build records are only kept up to this number of days'.

PICTURE 9. Jenkins job configuration for name, description and users.

Next SVN repositories were defined to the job. For the job, there is three different SVN locations where data, scripts and test cases will be checked out. Checkout operations were run using common credentials created just for the use of Jenkins. This is a common procedure to checkout files from SVN in an automation server.

First SVN repository contains mpp.properties file that has all the functions that are used to run all the steps required from this Jenkins job. Mpp.properties file is stored into root of the workspace of this job. Second SVN repository contains testing toolkit scripts which

are used in `mpp.properties` file to enable connectivity and file transfer between Jenkins and Windows clone test environment. Testing toolkit scripts were checked out into `libs` folder of this job workspace. Third SVN repository contains Robot Framework test cases, test data, and scripts to start test execution locally in Windows environment. These test cases, test data and scripts are stored in `tests` folder of the jobs workspace. Definitions of the SVN repositories can be seen in picture 10. The implementation of this third SVN location content is described in more detail in chapter 3.3.1 Suite structure.

Source Code Management

None
 Subversion

Modules

Repository URL	<input type="text" value="net.nokia.com/isource/svnroot/na_configurator/special/trunk/tools/M"/>	?
Credentials	<input type="text" value="cmci/*****"/> Add	
Local module directory	<input type="text" value="."/>	
Repository depth	<input type="text" value="infinity"/>	
Ignore externals	<input type="checkbox"/>	
		Delete
Repository URL	<input type="text" value="s/implementation/cmtat/implementation/RACITestingToolkit/src/mail"/>	?
Credentials	<input type="text" value="cmci/*****"/> Add	
Local module directory	<input type="text" value="libs"/>	
Repository depth	<input type="text" value="infinity"/>	
Ignore externals	<input type="checkbox"/>	
		Delete
Repository URL	<input type="text" value="ia.com/isource/svnroot/cm-teflon/CM4/REST-simulator/src/test/a1_1"/>	?
Credentials	<input type="text" value="cmci/*****"/> Add	
Local module directory	<input type="text" value="tests"/>	
Repository depth	<input type="text" value="infinity"/>	
Ignore externals	<input type="checkbox"/>	
		Delete

PICTURE 10. SVN repositories and checkout folders.

When project name, user credentials, SVN repositories, and checkout folders were defined the build section needed to be added to run the functions in `mpp.properties` file. For this Execute shell is used to run following commands: “`export MPP_DEBUG=1`” and

“.mpp/build”. These commands will start the execution of the steps defined in mpp.properties file. The functionalities in mpp.properties file will be gone through in the chapter 3.2.2.

Last step in the Jenkins job is post-build action to archive and publish Robot Framework test results. Directory is defined from where Robot Framework plugin gets the test results containing log and report files. After tests are run in the Windows environment the result files containing output.xml, log.html, report.html and screenshots are transferred into reports folder to be archived. Thresholds were defined for the tests so that yellow status is if 90% of critical tests are passed and green if 100% are passed. Execute shell and Robot Framework plugin configurations can be seen in picture 11.

The screenshot displays the Jenkins job configuration interface. It is divided into two main sections: 'Build' and 'Post-build Actions'.

Build Section:

- Execute shell:** A step with a command field containing the text:


```
export MPP_DEBUG=1
.mpp/build
```

 Below the command field is a link: "See the list of available environment variables". A red "Delete" button is located to the right of this step.

Post-build Actions Section:

- Publish Robot Framework test results:** A step with a text input field for "Directory of Robot output" containing the value "reports". Below this field is the text "Path to directory containing robot xml and html files (relative to build workspace)". To the right of the field is an "Advanced..." button. Below this are two input fields for "Thresholds for build result":
 - A yellow circle icon followed by a percentage sign and an input field containing "90.0".
 - A green circle icon followed by a percentage sign and an input field containing "100.0".
 Below these fields is a checked checkbox labeled "Use thresholds for critical tests only". A red "Delete" button is located to the right of this step.
- Archive the artifacts:** A step with a text input field for "Files to archive" containing the value "reports". A red "Delete" button is located to the right of this step.

PICTURE 11. Execute shell commands and Robot Framework plugin configurations.

3.2.2 Implementation and functions of the mpp.properties file

Mpp.properties file contains all the functions which are needed to run tests remotely on cloned Windows environment excluding SVN checkouts and test result visualization. Functions in this file contain higher level functionalities provided by different sources of MPP integration tools and low-level Linux commands.

The steps that were defined in the build() function in mpp.properties contains: removing reports folder, creation of local variable “return_value”, reserving Windows environment clone, resolving Windows environment clone’s network address, packing and copying test files into Windows environment clone, unpacking the test files and granting execute rights to test execution scripts, starting test execution on remote Windows environment clone, creation of reports folder and downloading test results from Windows environment clone to reports folder, release of the Windows environment clone. The mpp.properties file and build() function can be seen in the appendix 3.

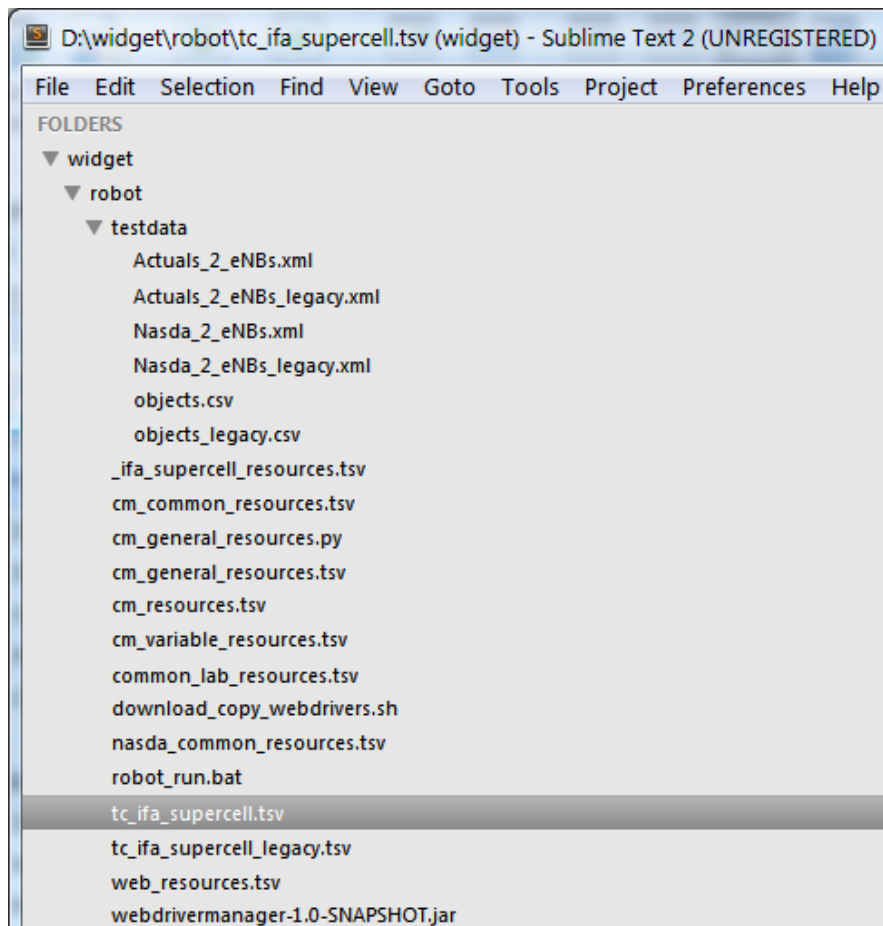
3.3 Test suite directory implementation

To be able to run end-to-end type of test cases against the application there are certain requirements from test suite directory. Test suite directory has resource files containing common keywords used to test web applications. To have some content in application there must be some test data also present to allow application function as required. Test suite directory also contains some scripts to start test run, and build jar-file that contains the functionality of WeDriverManager mentioned in chapter 3.1.2.

3.3.1 Suite structure

All needed test cases, resource files, test scripts, and test data were implemented based on per widget or application. The directory structure consists of robot folder where all above-mentioned files are. Usually the resource files containing combined keywords for specific use cases are stored in one centralized folder but this implementation was done this way to visualize all needed files in same folder and to perform as an example without

too much of complexity. Test suite specific test data was stored into same directory without parent directories indicating test data size like small, medium or large to keep directory structure simple for example application. Test data is XML or CSV formatted network element configuration data. The directory structure can be seen on picture 12.



PICTURE 12. Directory structure for widget test suite.

3.3.2 Files

tc_ifa_supercell.tsv and tc_ifa_supercell_legacy.tsv

These two files contain the test cases that are run against the widget or application. The test cases in these files are defined in very high-level keywords. Tests to be run are defined in these test case files based on tags in the beginning of the file. These files can be run in either Linux based system on top of Firefox web browser or in Windows environment on top of Google Chrome web browser.

cm_resources.tsv etc.

All tsv and python-files having resource in its name are used in the test case files. These files contain essential combinations of low-level keywords that can be used in the test case files to keep them simple. For example, resource files are used to import test data into SUT and preparation of test environment.

robot_run.bat and download_copy_webdrivers.sh

These two scripts were implemented to start test run and prepare the test environment. Download_copy_webdrivers.sh contains the command to download web drivers for each browser and copying those into Python27 folder. Last command in the download_copy_webdrivers.sh is to start test run by starting the robot_run.bat file. Robot_run.bat contains browser update functionality and python command to start the actual robot framework test execution. After the Robot Framework is finished all unnecessary tasks are killed in Windows test environment.

Test data

All test data is configuration data from actual network elements and are used in the testing of the widget or application. Test data is in the one folder in XML or in CSV format.

3.4 Test run launch possibilities

At this moment, the decision was made that the test run can be started by user any given time. Jenkins provides already an option to schedule test run when ever needed.

4 TEST EXECUTION AGAINST EXAMPLE APPLICATION

To verify the functionality of the test execution there should be some application running on top of web browser. The application chosen for this is web application which has been implemented about one year ago and has tests running against it in Linux test environment. In this chapter, the application is described and test execution results are demonstrated. On this step, the functionality was verified using Google Chrome web browser.

4.1 Description of example application

Application under example test run was an application running on top of Web browser called Supercell Tool. Application is used to configure a LTE Supercell which is a combination of maximum six sub-cells. The Supercell is a feature in LTE network that improves downlink performance in the sub-cells edge, improves uplink performance by dynamic selection of the best received sub-cell, and minimizes the need of handovers for example in high-speed trains. Supercell Tool application was necessary to easily configure the business sensitive Supercell functionality without causing any incorrect configuration which might cause network outage.

Supercell Tool application itself is somewhat simple. Application consists of possibility to select working set which contains – in this case – LTE network elements, Supercell creation mode, Managed Object table containing LTE network elements from working set, parameter table containing all needed info about parameter values from specific LTE cell, Saving the configuration as a plan into database, and possibility to provision the plan into network. All these components can be seen in the picture 13.

The screenshot displays the 'Plan supercell configuration for actual network elements' interface. At the top, it shows 'Working set: Tampere region' and 'Supercell creation mode: Combined supercell (LTE2445)'. Below this, there are input fields for 'Managed object DN' and 'Name'. A table lists several managed objects with their corresponding names. Below the table is a detailed configuration table with columns for LNCCEL ID, Name, SMOD/BBMOD, RMOD, RP3 port ID, Position in chain, LCELL, Actual supercell, Planned supercell, Actual subcell ID, Planned subcell ID, Actual subcell max power, Planned subcell max power, Actual merged cells, and Plan merge to. The table contains 10 rows of data for LNCCEL IDs 7901 through 7910. On the right side, there are sections for 'Save plan' (with a 'Name:' field and 'Save' button) and 'Run operations' (with a 'Provision' button).

Managed object DN	Name
PLMN-PLMN/RRBTS-2079	TRE_E15_A
PLMN-PLMN/RRBTS-2080	TRE_W25_C
PLMN-PLMN/RRBTS-137407	TRE_E40_B
PLMN-PLMN/RRBTS-140000	TRE_S20_A
PLMN-PLMN/RRBTS-140010	TRE_S20_B

LNCCEL ID	Name	SMOD/BBMOD	RMOD	RP3 port ID	Position in chain	LCELL	Actual supercell	Planned supercell	Actual subcell ID	Planned subcell ID	Actual subcell max power	Planned subcell max power	Actual merged cells	Plan merge to
7901		FSMF	FHEA	1	3	LCELL-1	Legacy		1				LCELL-1	-
7902	E15A_2	FSMF	FHEA	2	3	LCELL-2			2				LCELL-1	-
7903		FSMF	FHEA	3	2	LCELL-3	Normal							-
7904		FSMF	FHEA	6	2	LCELL-4	Normal							-
7905		FSMF	FHDB	1	2	LCELL-5	Normal							-
7906		FSMF	FHDB	2	2	LCELL-6	Normal							LCELL-3
7907		FSMF	FHDB	3	1	LCELL-7	Normal							LCELL-5
7908		FSMF	FHDB	6	1	LCELL-8	Normal							LCELL-6
7909		FSMF	FRGT	1	1	LCELL-9	Normal							LCELL-7
7910		FSMF	FRGT	2	1	LCELL-10	Normal							

PICTURE 13. Screenshot of Supercell Tool containing configuration data.

4.2 Content of the example test suite directory

Test suite directory structure is the same as described in the chapter 3.3 Test suite directory implementation. Section 3.3 described the basics of the suite content. This chapter will focus on the test cases in the .tsv-files. In the test suite directory, there is two different .tsv-files containing test cases to verify two different Supercell creation modes: Legacy supercell and Combined supercell. All test cases were run from these two .tsv-files using same tag when starting robot framework test run. In the appendix 4 the legacy supercell test cases can be seen in Robot Framework tab-separated values styled file. The test case file in appendix 4 is slightly modified so that there isn't any specific IP-address nor other sensitive information about internal test pipe solution. Test cases in the file are only in high-level keywords which are defined in the `_ifa_supercell_resources.tsv` file. This was done to keep actual test case files as simple as possible and the functionality in low-level is implemented into resource files.

In the test case file found in appendix 4, the target environment is hard coded by assigning specific IP-address. This is done to keep the implementation as simple as possible. The

IP-address also can be defined dynamically if this structure is used in a continuous integration pipe for some specific application production. This dynamic IP-address allocation will be gone through in the future use cases for this test pipe implementation.

4.2.1 Starting example test run on Windows environment

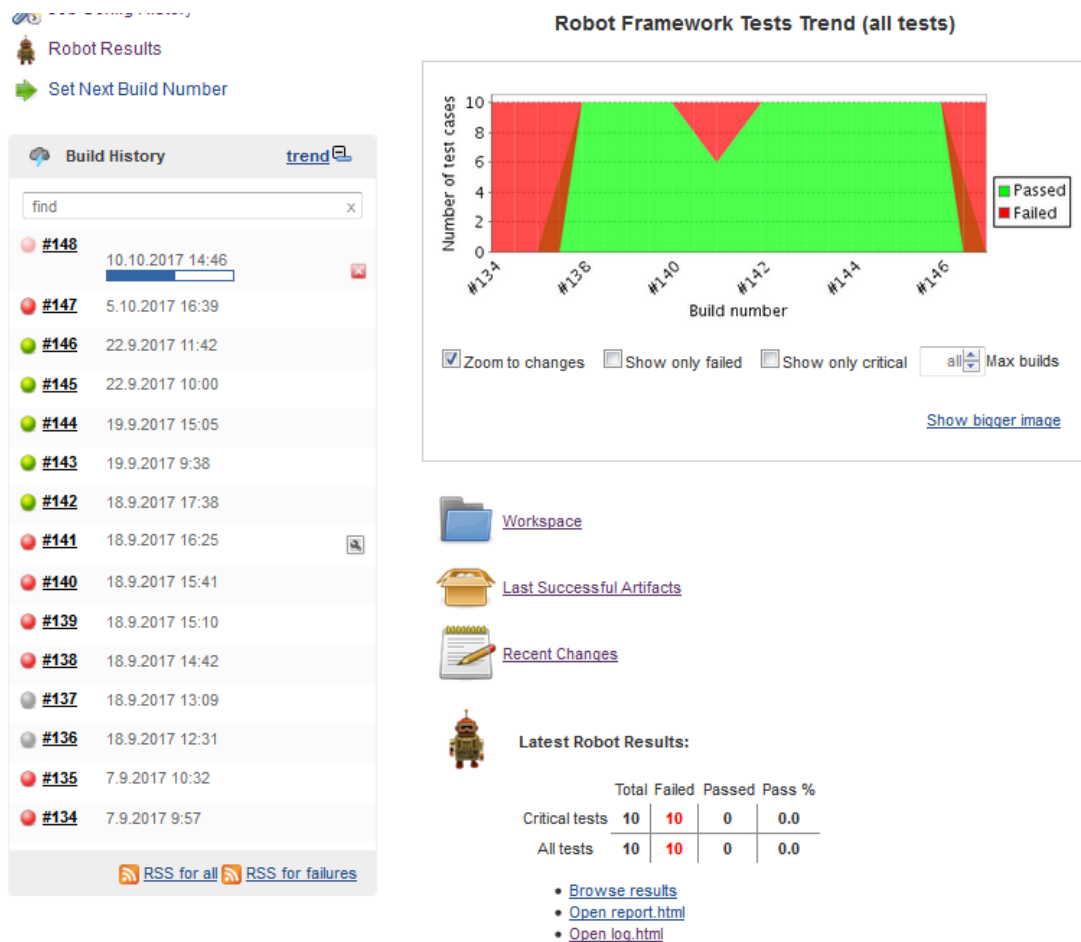
To start test case execution in remote Windows environment there must be some scripts to do this job. Shell script was run to download all web drivers, assigning execute rights to web drivers, and copying those into Python27 directory. At the end of shell script, a batch script was run which updated the Google Chrome and started the Robot Framework test execution. The content of `download_copy_webdrivers.sh` shell script can be seen in appendix 5. Implementation of update procedure, Robot Framework test execution start and termination of `java.exe` and `cmd.exe` can be seen in appendix 6. `robot_run.bat`.

4.3 Test result and visualization

After the test cases were run Robot Framework provided a great test report and log containing crucial screenshots of the test run. In this case two test case files contained 10 test cases in total. Robot framework produced `output.xml` file from the test run and based on that it generated test case report which is a high-level presentation of passed and failed testcases. Generated log-file contains all keywords which are used in test case execution in details. From these files Jenkins can show test result trend and other information on the Jenkins job page.

4.3.1 Test result view on Jenkins job page

High level test results can be seen on the Jenkins job page for the latest build. Build history and the result of each test run can be seen on the left-hand side of the page. Test results are published by Robot Framework plugin configured for this job. On the Jenkins job page Robot Framework Tests Trend, Latest Robot Results, and links to latest results are shown and can be seen in the picture 14.



PICTURE 14. Jenkins Job page containing Robot Framework test result visualization.

From the view in the picture 14, end-user can see that latest test run #147 have all test cases in failed state. Also, it is visible that new test run has started with build number #148. On the Jenkin job page links are provided for the latest test run to allow end-user quick access to test results. Access to the latest results straight from Jenkins job page helps end-user quickly find possible software bugs and report those via fault management tool.

4.3.2 Test report and log html-files

After every Robot Framework run these two html-files are generated so individuals can inspect the test results. These files are report.html and log.html files. In the picture 15, report html can be seen. In the report, the pass / fail ratio can be seen in a high-level. Report provides statistics based on tags, by suite and in total. Report page contains links that forwards the user into log.html if further investigation of failed test cases is needed.

In Test Details section of report page user can define if he or she wants to list only critical or all tests. After defining type of the test more specific separation of failed and passed test cases can be seen in a report html-page. Report html-page will have red background if even one of the critical test cases fails and correspondingly the background will be green when there are no failed critical test cases.

Tc Ifa Supercell Legacy & Tc Ifa Supercell Test Report

LOG Generated: 20170918 16:41:46 GMT +03:00
1 day 2 hours ago

Summary Information

Status: **4 critical tests failed**
 Start Time: 20170918 16:31:32.538
 End Time: 20170918 16:41:45.946
 Elapsed Time: 00:10:13.408
 Log File: log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	10	6	4	00:04:43	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>
All Tests	10	6	4	00:04:43	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
CM2	10	6	4	00:04:43	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>
GUI_Web	10	6	4	00:04:43	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>
IFA_SUPERCELL	10	6	4	00:04:43	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>
nadcproto	10	6	4	00:04:43	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Tc Ifa Supercell Legacy & Tc Ifa Supercell	10	6	4	00:10:13	<div style="width: 60%; background-color: green;"></div> <div style="width: 40%; background-color: red;"></div>
To Ibs Supercell Legacy & To Ibs Supercell. Tc Ifa Supercell Legacy	2	2	0	00:04:57	<div style="width: 100%; background-color: green;"></div>
To Ibs Supercell Legacy & To Ibs Supercell. Tc Ifa Supercell	8	4	4	00:05:17	<div style="width: 50%; background-color: green;"></div> <div style="width: 50%; background-color: red;"></div>

Test Details

Totals Tags Suites Search

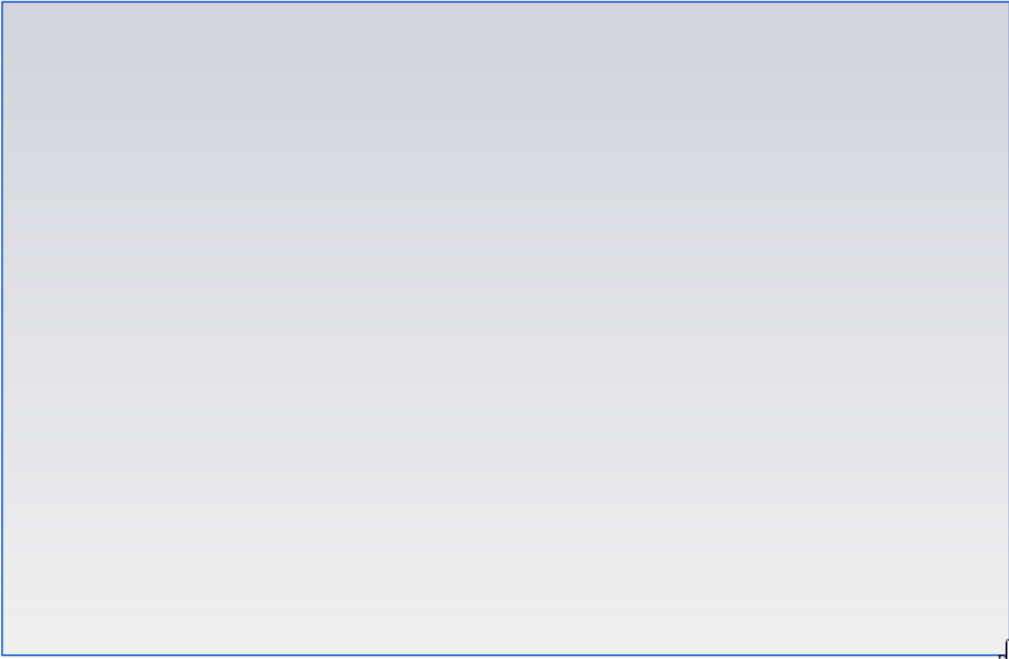
Type: Critical Tests All Tests

PICTURE 15. Report html-page containing passed and failed test cases.

In the log html-page user can go deeper into details to see in which test case step or keyword the test failed. In the picture 16 the failed test case step / keyword can be identified. After the keyword fails the screenshot can be taken and integrated into log html page. Screenshot keyword is not integrated into syntax when test case fails, but should be added into test cases. This will help the user to pinpoint the exact step in which the test case failed. In picture 16, can be seen the situation where the page is completely empty and the frame can't be selected. This will set the keyword as a failed state.

KEYWORD seleniumLibrary.Select Frame \${solfraemId}
Documentation: Sets frame identified by 'locator' as current frame.
Start / End / Elapsed: 20170918 16:40:02.046 / 20170918 16:40:02.288 / 00:00:00.242

KEYWORD seleniumLibrary.Capture Page Screenshot
Documentation: Takes a screenshot of the current page and embeds it into the log.
Start / End / Elapsed: 20170918 16:40:02.062 / 20170918 16:40:02.288 / 00:00:00.228
16:40:02.288 INFO



16:40:02.046 INFO Selecting frame '//*[@id="super-cell-widget"]'.
16:40:02.288 FAIL ValueError: Element locator '//*[@id="super-cell-widget"]' did not match any elements.

PICTURE 16. Failed keyword in log.html file

5 FEEDBACK AND ANALYTICAL COMPARISON WITH PREVIOUS TEST SETUP

In this chapter comparison will be made between new test execution solution for web browser based applications and the old way. Focus in this chapter is the comparison between Windows environment and Linux environment. In addition of comparison the challenges and benefits of this project will be analyzed. Also, some feedback will be documented here from SW architect and engineers running tests using the new solution for Windows environment test execution.

5.1 Benefits of new solution

There were numerous benefits known in advance of this solution which were the key factors to start this project. All testing done in Windows environment for Web based applications have been executed manually before this solution. In the long run, this solution will free resources from manual testing to other value-added work. Following chapters will go through main benefits of this solution per each technical entity.

5.1.1 Test execution environment

Prior to this project all Web based application testing have been executed automatically on top of Firefox running in Linux OS. The old solution doesn't really cover end-user like environment fully, when Linux and Windows operating systems are compared by division of usage. Windows is the most common OS installed on end-user client PC's. To really test end-to-end functionality of web based applications, it's crucial to verify the functionality of the developed application running on most common web browser provided for Windows OS. Most of the time web applications work similarly in all common web browser, but there are occurrences when browser update or development targeted to one browser has broken the functionality in other web browsers. On the short period of time this new solution has already caught some major faults in one browser when the functionality has been normal in other web browsers. Introducing the end-user-like environment for test automation will increase quality of the developed web applications to another level. This will also give insights to web developers to get fast feedback loop,

which allows them to learn the typical problem areas of each common web browser used in Windows OS.

Linux test execution environments are running on dedicated virtual machines in the same network where products are tested. The new Windows environments are cloned by request from one Windows template image. After the cloning request one virtual machine is reserved and Windows environment is ready to be used in seconds. This way of providing Windows environments allows data center infra to respond to the testing needs in a flexible way. This approach will scale easily to the needs of testing if there will be a need for multiple simultaneous test executions at the same time. Resource usage is also minimal when there are no tests running, since every virtual machine is released after tests are run in Windows environment.

5.1.2 Test execution orchestration

Test execution orchestration is configured to be an independent testing job in the Jenkins. Configuration is kept relatively simple, which will ease the duplicating the job and use the configuration for another web application testing with minor changes. This will benefit the web application testing in the future when new web applications are productized and ready to be implemented. When this test automation solution is ready in the beginning of web application breakthrough for the product, it will alleviate the tasks to create automated end-to-end testing solution for other upcoming web applications.

Benefit of the simple Jenkins job is that it can be easily added as a quality step into product continuous integration pipe. This type of Jenkins testing job can be added as one of the last steps of continuous integration pipe after smoke tests. Introducing this testing job into regression purposes will also give value into quality.

5.1.3 Test suite directory

The whole test suite directory has accumulated into disorderly structure of libraries, test data, resources and test case files. All test related files are in one big directory structure containing both web application and Java desktop application files. This is not feasible

since both application types share little common resources and test data. Now when new tests are created to be run against browsers running on top of Windows OS, all resources and test data that are in the directory structure is not needed. This gave the opportunity to specify the structure of test suite directory from scratch and leave out all unnecessary files. Benefit to this re-planning of test suite directory is to provide simple and easy to understand structure to store testing related files. Now when all web applications have their own folder, all specific resources and test data can be added into correct place. This solution also provides the structure so that the parent folder can contain shared resources for all web applications.

5.2 Challenges in the new solution

This solution has added relatively much maintenance load and some of it have been hard to automate. Most of the challenges were encountered during Windows environment configuration and automating certain function like updating the web browsers before test execution. Some other challenges included communication and session management issues between Linux and Windows systems. During the implementation of this solution, it has come clear that Windows environments are not that popular for automated testing purposes, because of the lack of possibilities to easily automate majority of test execution environment related tasks. Following chapters will go through main challenges of this solution per each technical entity.

5.2.1 Test execution environment

From the experience of this implementation it seems that Windows as a test execution environment is not easy to maintain. When the clone of the Windows environment is started on virtual machine, first thing is to check if web browser should be updated. Today web browsers update themselves many times per month. Updating web browsers in the beginning of each test execution adds more time to the test execution. Browser update can be done by starting update executable using scripting and then it must be monitored when it's ready. When all major browsers have different ways how they are updated, it produces more complexity to maintaining Windows environment.

Apart from browsers Windows OS also requires critical security updates time to time. These updates can't be done in the beginning of the test execution, because Windows OS updates might need restarts or take too much time. This issue can be solved by having two parent Windows environments where other can be updated accordingly and another is used for cloning purposes. This is very time consuming and precise work that need to be done manually time to time. The whole network might become under a risk of virus attacks if Windows OS updates are neglected.

One major challenge is the forced resolution when connecting into Windows environment via SSH provided by Cygwin OpenSSH-service. When the connection is established from Jenkins via SSH into Windows the opened session doesn't have any rights for Windows environment graphic card settings to change the resolution. This is called session 0 which is isolated session to mitigate security risks in the Windows OS (Application Compatibility: Session 0 Isolation, 2017). The resolution in this case is forced to be 1024 * 768 pixels. This won't prevent the testing in Windows environment, but gives some restrictions to test case planning. This resolution issue should be taken into consideration on the next steps of enhancing the current solution.

5.2.2 Test execution orchestration

Test orchestration using Jenkins is mostly straight forward process, but it includes many different own entities. All these entities must be online and reachable every time when tests are run. These entities are for example software revision control system, Jenkins itself and scripts needed for connectivity related tasks. Connectivity scripts have already caused some problems when they needed to be changed to add more security into Linux-Linux SSH connections. Due to these changes, same scripts couldn't be used for Linux-Windows SSH connections, but new scripts needed to be implemented.

Since Jenkins is not main resource to find out test results for whole product component, it might be hard to link the test results into already existing solution where end-to-end test automation results are gathered. In the future if more and more test jobs are added into Jenkins there should be some central place to store all test results from different web based applications.

5.3 End user feedback about test pipe

In this chapter feedback from SW architect and SW engineer are taken into closer look. SW architect has been main contributor demanding new way of testing web based applications. SW engineer also has given feedback about the solution and what were the positive changes.

5.3.1 Feedback from SW architect

From the SW architect point of view, this solution provides the way how quality assurance must be done in modern DevOps software development mode. It is also emphasized that all web browser variants running on different versions of Windows OS can't be regression tested manually with current resources. It was also noted that Windows is most commonly used OS in telecommunications sector, which compels us to concentrate on the quality assurance as close as possible with end-user like environment.

SW architect summarizes that this has been a project that will enable our department to be ready for the future challenges what continuous deployment and DevOps will require. Full email from SW architect can be seen in appendix 7.

5.3.2 Feedback from SW engineer

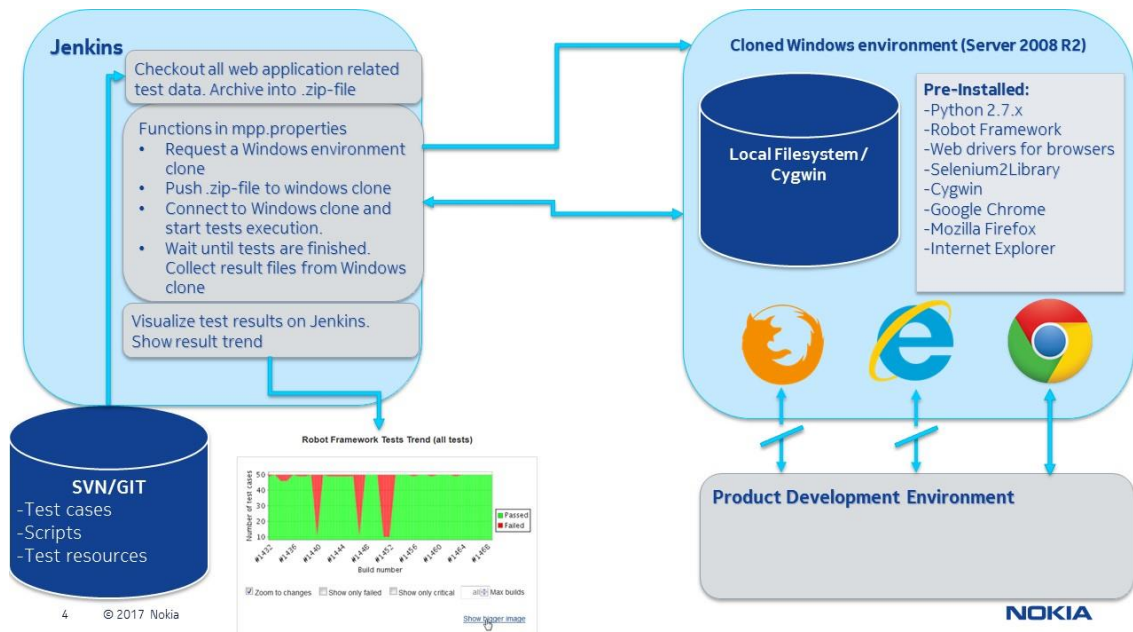
SW engineer also emphasizes the real need for end-user like environment where test automation can be run. Already before Windows environment test automation SW engineer has found faults from the functionality of a web based applications. These faults have been found from one browser but web application has been working faultlessly in other browsers. Some minor faults have been found from same browser between different browser versions.

From SW engineers point of view, this will allow quick and easy way to test the functionality of the developed code. It's also seen that this solution will keep the quality high on the product when the solution is used in regression testing purposes. SW engineer has

also point that there is still much to add into this solution to provide test execution environments to all different end-user like environments. Email review and feedback from the project can be seen in appendix 8.

5.4 Project outcome compared to high-level requirement description

Based on original high-level requirement the implementation of this project did not change the original idea notably. Main differences between original high-level requirement and actual outcome of this project were: Test execution orchestration on Jenkins was done by scripting and using mpp.properties file, and not all web browsers were tested to avoid duplicity. In the picture 17, the high-level illustration of completed project can be seen. Firefox and Internet Explorer are kept in the picture since they are installed into Windows environment and related web drivers can be updated, even though they are not tested in the scope of this project. Scripting needs were bigger than though in the beginning of this project. Scripting needs to update Google Chrome and web drivers gave a lot of challenges during the project even though those are not visible in the high-level requirements.



PICTURE 17. Illustration of complete project in a high-level.

5.5 Reasoning

Even though the maintenance load is relatively high on Windows environment setups, the solution is a quality enabler for web based application continuous integration pipe. Automation of Windows environment testing will become even more important when production is moving towards continuous delivery mode. Already before the implementation of test automation solution for Windows environments, many issues were found in manual testing. These issues would have caught with sufficient test automation on Windows environment. When this solution is taken into use in wider scope the benefits will multiply and manual testing resources can be released to value added work.

Creating test data and running same test cases similarly for example three different web browsers is tedious and time consuming. This kind of manual testing should be done whenever web browser is updated or minor change is done to verify that nothing gets broken. To automate this end-to-end testing, the feedback loop to developer is much faster. Today when almost all software businesses are moving towards continuous delivery it is crucial to automate as much as possible. This allows verification engineers to test there where it is really needed.

6 FUTURE USE CASES

After introducing this solution that provides Windows environment where end-to-end test automation can be run, there are many different steps in the product development where this solution can be included. Most reasonable place to include the Windows environment testing is regression testing phase. This solution frame can also be taken into use by any other project developing their own web applications. This can be easily promoted within the company as an end-to-end automated testing solution. In the near future, there might be reasonable to evaluate the introduction of similar kind of Windows environment test automation solution using containers. Creating Windows environment for test automation brought up discussion about automated testing possibilities on macOS.

6.1 Next steps to get full potential out of the Windows environment

This project was implemented so that the whole integration pipe is ready, but the Windows environment and test execution on all major browsers require more implementation. Functionality of this solution was verified using Google Chrome. Next steps will be setting up support for Mozilla Firefox and Microsoft Internet Explorer. Both browsers need to be able to be updated in automated way in the beginning of test execution like as Google Chrome. When all browsers can be updated automatically and test cases run on each of them, then quality of the applications will be guaranteed.

Security aspects and Windows updates are something that should be taken under investigation before the usage of this solution spreads further in the company. Usage of Microsoft Active Directory should be considered as an option to update Windows machines within reasonable intervals. If costs are too high for Active Directory, then some other way should be implemented to maintain Windows environment security.

6.2 Windows environment test automation on different verification steps

In this chapter, some possible verification steps are introduced where this Windows environment test automation solution can be integrated. Two main places to introduce this

solution are daily regression step and QL4 (Quality Level 4) testing step. These two testing steps are orchestrated by Jenkins, which allows adding the dependency for Windows environment test automation step. Both QL4 and daily regression testing steps must have an environment where latest system build of the component is upgraded. After either one of the mentioned test steps are run, Windows environment test cases can be run against that same build to verify the functionality of the applications running on Windows OS web browsers. IP-address for the environment where latest system component build is upgraded, must be obtained dynamically. There is automation already for getting IP-address and can be used into Windows environment test cases via resource file got from Jenkins. IP-address can be get from the resource file by using same variable name in the Robot Framework test case file.

6.2.1 Daily regression testing step

Daily regression step is testing phase where big number of automated test cases are run against system component build once a day and usually takes many hours to complete. Daily regression is not a quality gate for promoting the system component forward into product because of the lengthy execution time, but it gives valuable information to developers if some minor faults are found. This is the logical first step where this Windows environment test automation solution can be integrated. Integration to daily regression gives possibility to enhance the stability of the Windows environment test automation process. After stability is confirmed in the daily regression step, the Windows environment test automation can be integrated into QL4 test step.

6.2.2 QL4 testing step

QL4 testing refers to a step where the functionality of our system component is verified before it is promoted forward into product. If QL4 testing fails the system component build won't be promoted further and system component build is regarded as faulty. Adding the Windows environment testing into QL4 testing step requires that all parts from Windows environment to the Jenkins job orchestration functioning properly. Proper functionality is needed because QL4 testing step shouldn't have any test cases failing because of the unstable testing setup. When adding the Windows environment testing into QL4

testing step the test case content must be reviewed so that executed test cases will not add too much time into feedback loop.

6.3 Front end development possibilities

In addition of test automation execution, the front-end developer could use Windows environment to quickly verify that his or her application UI (User Interface) works in all major Windows web browsers. This would require implementation of simple backend simulator with API (Application Programming Interface) which will enable test data transfer between web application and backend. Web application itself could be deployed in a way that it is running in localhost. When web application is deployed it could be tested automatically on major browsers. This would shorten the feedback loop even more and prevents front-end developer committing faulty code into version control system.

6.4 Other applications outside of current development area

This solution is easy to take into use if some other system components have Jenkins in use. Within our product there are many other already existing applications running on web browser and the testing can be automated with this solution. To enhance product quality the all system component should have Windows environment test automation solution in their continuous integration pipe as a quality step. In this way, the product would be much more mature and extensive manual regression testing could be mitigated a lot.

Test automation solution for Windows environment can be introduced into System level verification. This is the step where the main functionalities are tested against fully built product installed into customer-like environment. There are already some level automated testing and web application test automation on top of Windows environment could cut time of from manual testing.

6.5 Macintosh operating system

Usage of Macintosh operating system – later macOS – usage should be surveyed to determine if web application should be tested on main web browsers supported by macOS. If there is a need to verify web applications in macOS environment, this solution could be used as a reference to create own test automation solution for macOS environment. Since macOS is based on Linux-like system the communication with Jenkins and macOS environment would be much simpler. In the other hand, there could be expensive license fees if macOS would be used in multiple virtual machines simultaneously.

6.6 Windows web browser testing and docker containers

Now containers are coming more and more common in the SW development work. In the future, there could be possibility to run each web browser in own container. This would enable simultaneous test execution for wanted web browsers. Containers could ease the security problems compared to cloning the Windows environment from snapshot. The Windows environment could be updated continuously without disturbance of web browsers running in containers. Containers are more light weight than virtual machines and start time is even faster.

Containers might have some challenges and there should be some comparison between challenges and benefits. The resource usage might be challenge in container usage since containers are running on host OS. This would mean that host should have adequate resources to accommodate multiple containers running on the same time. If the infrastructure regarding Windows OS can't be scaled to respond the resource needs, there might be unused resources outside of the peak hours. Now there is minimal amount of information available for container usage in Windows OS, which will add more challenges into container introduction to test automation solution for Windows environment.

REFERENCES

Jenkins. Main page

Read 30.10.2017. <https://jenkins.io/>

VMware vSphere. "Learn More about vSphere" – video.

Watched 22.03.2017. <http://www.vmware.com/products/vsphere.html>

Apache Subversion. "Enterprise-class centralized version control for the masses".

Read 22.03.2017. <https://subversion.apache.org/>

Robot Framework. Introduction.

Read 22.03.2017 <http://robotframework.org/>

Downloads Python | Python.org

Read 22.03.2017 <https://www.python.org/downloads/>

How To Get SSH Command-Line Access to Windows 7 Using Cygwin,

Read 03.07.2017 <https://www.howtogeek.com/howto/41560/how-to-get-ssh-command-line-access-to-windows-7-using-cygwin/>

Java SE Runtime Environment 8 Downloads,

Read 03.07.2017 <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

Jenkins Tutorial,

Read 20.04.2017 https://www.tutorialspoint.com/jenkins/jenkins_pdf_version.htm

Application Compatibility: Session 0 Isolation,

Read 24.10.2017 <https://msdn.microsoft.com/en-us/library/bb756986.aspx>

APPENDICES

Appendix 1. TestRunner.java

```

// Copyright (C) 2017 Nokia Solutions and Networks. All rights reserved.
// include webdrivermanager package and import from github project wdm
package webdrivermanager;
import io.github.bonigarcia.wdm;

public class TestRunner {
    // Set PROXY
    private static final String PROXY = "demuprx-fiesprx.glb.nsn-net.net:8080";
    //main function
    public static void main(String[] args)
    {
        // get browser as an argument and give error if empty
        String browser = args.length!=0 ? args[0] : "No argument given!";
        switch (browser) {
            //download chrome driver if argument is gc
            case "gc":
                setUpManager(ChromeDriverManager.getInstance());
                break;
            //download ie driver if argument is ie
            case "ie":
                setUpManager(InternetExplorerDriverManager.getInstance());
                break;
            // download firefox driver if argument is ff
            case "ff":
                setUpManager(FirefoxDriverManager.getInstance());
                break;
            // download all browser drivers if argument is all
            case "all":
                setUpManager(FirefoxDriverManager.getInstance());
                setUpManager(InternetExplorerDriverManager.getInstance());
                setUpManager(ChromeDriverManager.getInstance());
                break;
            // give error message if argument is not recognized
            default:
                System.out.println("Unsupported browser or unknown argument: " +
                    browser );
                break;
        }
    }
    // Used in switch case to set proxy and setup
    private static void setUpManager(BrowserManager manager) {
        manager.proxy(PROXY);
        manager.setup();
    }
}

```


Appendix 2. pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://ma-
ven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.nokia.oss.configurator.webdrivermanager</groupId>
  <artifactId>webdrivermanager</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
  <dependency>
    <groupId>io.github.bonigarcia</groupId>
    <artifactId>webdrivermanager</artifactId>
    <version>1.6.2</version>
  </dependency>
</dependencies>

  <build>
  <plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
    <archive>
    <manifest>
      <addClasspath>>false</addClasspath>
      <mainClass>webdrivermanager.TestRunner</mainClass>
      <addDefaultImplementationEntries>>true</addDefaultImplementationEntries>
    </manifest>
    </archive>
    </configuration>
  </plugin>

  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>

    <configuration>
    <createDependencyReducedPom>>true</createDependencyReducedPom>
    <filters>
    <filter>
      <artifact>*:*</artifact>
      <excludes>
        <exclude>META-INF/*.SF</exclude>
        <exclude>META-INF/*.DSA</exclude>
        <exclude>META-INF/*.RSA</exclude>
      </excludes>
    </filter>

```

```
</filters>
</configuration>

<executions>
  <execution>
    <phase>package</phase>
    <goals>
      <goal>shade</goal>
    </goals>
    <configuration>
      <transformers>
        <transformer
          implementation="org.apache.maven.plugins.shade.resource.ServicesRe-
sourceTransformer" />
        <transformer
          implementation="org.apache.maven.plugins.shade.resource.ManifestRe-
sourceTransformer">
          <manifestEntries>
            <Main-Class>webdrivermanager.TestRunner</Main-Class>
          </manifestEntries>
        </transformer>
      </transformers>
    </configuration>
  </execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

Appendix 3. mpp.properties

```

[[ "$(whoami)" == "cmci" ]] && mpp_svn="$mpp_svn --config-dir /home/cmci/.subver-
sion_mpp"

# Used sources
source .res/mpp_clone_client || fail "Unable to load mpp clone client ($?)"
source .cpp/build-tools

infrastructural_build_issue_handler=0

# User name used for clone reservation
res_user="cm"

# Reservation ID resolved using Job name and build number of the job
res_id="${JOB_NAME}_${BUILD_NUMBER}"

# Reservaton type Windows Server R2
res_type="rcm_win_2008_r2"

# Maximum duration of Windows test environment clone reservation in minutes
res_duration=20

backend_project="cm"
mpp_ris_id=""

# Windows test environment password for Administrator user
password=Passw0rd

# Main function that contains Windows test environment clone lab reservation, copying
files to
# Windows test environment clone, unpacking the test files and starting test execution in
# Windows test environment clone, retrieving test results from Windows test evinronment
clone,
# and releasing the Windows test environment clone
function build()
{
# Remove reports folder from Jenkins job workspace.
rm -rf reports robot
# Local return calue set to 0
local return_value=0;
reserve_lab
server=${res_clone_id}node01.netact.nsn-rdnet.net
copy_files_to_server || { release_reservation; return 1; }
unpack_and_run_tests_on_server || return_value=1
get_test_report || { release_reservation; return 1; }
release_reservation || true
return $return_value
}

# puts function uses puts script with user Administaror and $password and $@ as a argu-
ment.

```

```

function puts()
{
  libs/puts Administrator@$server $password $@
}

# gets function uses gets script with user Administrator and $password and $@ as a argument.
function gets_win()
{
  libs/gets_win Administrator@$server $password $@
}

# ssh function uses ssh script with user Administrator and $password and $@ as a argument.
function ssh()
{
  libs/ssh Administrator@$server $password $@
}

# Clears the home folder of the Administrator in Windows test environment clone,
# creates tarball of the test files and scripts,
# and transfers the tests.tgz tarball to the Windows test environment clone.
function copy_files_to_server()
{
  ssh rm -f home/Administrator/*
  tar czf tests.tgz -C tests --exclude=.svn .
  puts tests.tgz home/Administrator
}

# uses ssh() function to extract tests.tgz,
# granting run rights to test execution scripts and starting the tests
# in Windows test environment clone.
function unpack_and_run_tests_on_server()
{
  ssh tar xzf tests.tgz
  ssh chmod +x download_copy_webdrivers.sh
  ssh chmod +x robot_run.bat
  ssh /home/Administrator/download_copy_webdrivers.sh
}

# uses gets() function to retrieve the test results from
# Windows test environment clone containing all .png screenshots.
function get_test_report()
{
  mkdir reports
  gets_win home/Administrator/output.xml reports
  gets_win home/Administrator/log.html reports
  gets_win home/Administrator/report.html reports
  gets_win "home/Administrator/*.png" reports || true
}

```

```
# Reserves the Windows test environment clone
function reserve_lab()
{
  init_res_build_token
  init_reservation_parameters
  init_res_max_queue_time
  local res_max_queue_time=0
  local polling_interval=10
  while true
  do
    queue_for_clone_win && return
    [[ $? -eq 1 ]] && return 1
    sleep "$polling_interval"
  done
}

# uses queing mechanism provided by mpp integration tools.
function queue_for_clone_win()
{
  res_service="$res_service_clone_pools"
  queue_for_reservation
}
}
```

Appendix 4. tc_ifa_supercell_legacy.tsv test case file

Settings

Documentation End to end regression tests for IFA (Intelligent Feature Activation) Supercell feature.
 Suite Setup Setup_Suite
 Suite Teardown Cleanup_Suite
 Force Tags IFA_SUPERCELL CM2 GUI_Web nadcproto
 Library String
 Library BuiltIn
 Resource _ifa_supercell_resources.tsv

Variable

#Added variables

\${SDV_WEBSPIHERE_APPSERVER_PRIMARY_IP} 0.0.0.0
 \${SDV_DESKTOP_ADMIN_PASSWORD} password

Test Cases

IFA_Supercell_Create_Legacy
 [Documentation] MRBTS-7407 merge LCELL-8 to LCELL-0
 IFA_Supercell_Create_Legacy_test

IFA_Supercell_Deactivate_Legacy
 [Documentation] MRBTS-7407 remove LCELL-5 from LCELL-12
 IFA_Supercell_Deactivate_Legacy_test

Keywords

Setup_Suite [Timeout] 3 minutes
 common_lab_resources.Open Lab \${SDV_WEBSPIHERE_APPS-
 ERVER_PRIMARY_IP} \${SDV_DESKTOP_ADMIN_PASSWORD}
 Setup Test Data Into Lab Legacy
 Open Start Page
 Start IFA

Cleanup_Suite [Timeout] 5 minuts
 Selenium2Library.Close All Browsers
 Cleanup Testdata From Lab
 SSHLibrary.Close Connection

Appendix 5. download_copy_webdrivers.sh

```
#!/bin/sh
#download web drivers for Internet Explorer, Google Chrome and Firefox
java -jar webdrivermanager-1.0-SNAPSHOT.jar ie
java -jar webdrivermanager-1.0-SNAPSHOT.jar ff
java -jar webdrivermanager-1.0-SNAPSHOT.jar gc
#grant execute rights to webdrivers
chmod 755 /cygdrive/c/Users/Administrator/.m2/repository/webdriver/chromedriver/*
chmod 755 /cygdrive/c/Users/Administrator/.m2/repository/webdriver/geckodriver/*
chmod 755 /cygdrive/c/Users/Administrator/.m2/repository/webdriver/IEDriverServer/*
#copy webdrivers into Python27 directory so they are in PATH
find /cygdrive/c/Users/Administrator/.m2/repository/webdriver/ -perm /a+x -exec cp {}
/cygdrive/c/Python27 \;
#start batch script which contains google chrome update and Robot Framework test case
execution
./robot_run.bat
```

Appendix 6. robot_run.bat

```
Start "" "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
sleep 5

set count=2
:while
  tasklist /FI "IMAGENAME eq GoogleUpdate.exe" /FO CSV |grep GoogleUpdate |WC
  -l >result.txt
  set /p index=<result.txt
  echo The value of index is %index%
  sleep 5
  if %index% geq %count% (
    goto :while
  )
del result.txt
taskkill /IM chrome.exe

python -m robot --include IFA_SUPERCELL tc_ifa_supercell_legacy.tsv tc_ifa_super-
cell.tsv
taskkill /F /IM java.exe
taskkill /F /IM cmd.exe
exit /B 0;
```


Appendix 7. Feedback from SW architect

In DevOps software development mode unit testing and test automation are the way how quality assurance and verification must be done. This applies even in the customer deliveries in case of the advanced customers hosting they own environment or ones relying vendor provided cloud solution. There cannot be constant manual regression test phase simply because cycle of the change is so fast that even with vast resources it cannot be handled traditional way.

Microsoft Windows is the most popular operating system used for running Web desktop applications specially in telecommunication sector. Several variants of the OS as Windows 7 and 10 bring new challenges for the testing because all possible browser variants e.g. Chrome, Firefox, IE and Edge should be possible to test. Also constantly updated browser versions must be automatically handled by the solution thus it is almost impossible and waste of resources to do such thing manually.

Microsoft Windows Web UI test automation developed by Rami Lehtelä has solved all of these complex problems using modern software capabilities. This work is enabler for DevOps and building advanced CI and verification pipe for our department.

- Mikko

Appendix 8. Feedback from SW engineer

There has been a real need for Windows test automation (TA) solution, and this work has been done to fill the needs. Earlier we have had only TA solution using Linux and only one browser, while majority of the end users use Windows operating system and browsers in it. We have already seen that some faults exist in some browser while other browsers work fine. With this Windows TA solution we have a possibility to test using different operating systems, different browsers and latest browser versions. This improves the quality and reliability of our testing, and because of this end users should face less problems and improve the user experience.

This Windows test automation solution is also good tool for developers, developers can do their coding and after that easily test with different browsers automatically. This is even more important in maintenance phase, when some bug fixing / code changes are done, that there is easy way to check that existing functionality still works.

There are still work to do for example we maybe need to test with different Windows versions and cover even more browsers, but this has been a good starting point and improvement.

-Sari