OULU UNIVERSITY OF APPLIED SCIENCES

Elvis Punnek

AUTOMATION FOR A BASE STATION STABILITY TESTING

AUTOMATION FOR A BASE STATION STABILITY TESTING

Elvis Punnek
Bachelor’s Thesis
Spring 2016
Information Technology
Oulu University of Applied Sciences

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Langattomat laitteet

Tekijä: Elvis Punnek
Opinnäytetyön nimi: Tukiaseman stabiliteettitestauksen automatisointi
Työn ohjaajat: Kari Jyrkkä, Antti Seppälä, Pete Pietilä
Työn valmistumislukukausi ja -vuosi: Kevät 2016, Sivumäärä 38 + 1 liitettä

Tämän opinnäytetyön tilasi Oy LM Ericsson AB Oulu. Työ tilattiin LTE tukiaseman stabiiliuden testiautomaatioratkaisun tutkimisen ja toteuttamisen helpottamiseksi.

Työn päätavoite oli toteuttaa testiautomaatio etukäteen suunnitellulle testisarjalle. Tämä testiautomaatioratkaisu haluttiin toteuttaa tietylle ympäristölle ja laitteistolle. Työ sisälsi automaation toteutuksen testitapauksille sekä niiden integroinnin päivittäiseen automaatio ajoon.

Työn tärkein osa-alue oli testisarjan toteutus, joka sisälsi useita testitapauksia. Kaikkien testitapauksien automaatio oli tarpeellista, jotta manuaalista konfigurointia ei tarvitsisi tehdä päivittäin. Nämä testitapaukset sisälsivät osia ympäristön alustamisesta, testisarjojen toteuttamisesta sekä sen jälkeen, testitulosten raportoinnista. Pääasiallinen testiautomaatioympäristö oli rakennettu Java -pohjaisen ratkaisun päälle, joten tämän vuoksi suurin osa uudesta toiminnallisuudesta myös toteutettiin Java -ohjelmointikielellä.

Testiautomaatioratkaisu toteutettiin onnistuneesti ja laitettiin päivittäiseen testiautomaatioajoon. Testiautomaatioon toteutettuja testitapauksia tullaan jatkokehittämään tulevaisuudessa uusien ominaisuuksien tai korjaustarpeiden ilmaantuessa. Testiautomaation testiraportointiosa raportoi kaiken tarvittavan informaation testien ajon jälkeen ja tätä voidaan käyttää käytössä olevan ohjelmiston tilan toteamiseen.

Testiautomaatioratkaisua käytetään päivittäin testiautomaatiolaitteissa ja se tarjoaa paljon tietoa viimeisestä ohjelmistoversiosta. Ratkaisu myös vapauttaa huomattavan määrän aikaa manuaalisesta testauksesta muihin tarpeellisiin asioihin.

Asiasanat: LTE, Testiautomaatio, Java

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Wireless Devices

Author: Elvis Punnek
Title of the bachelor’s thesis: Automation for a base station stability testing
Supervisors: Kari Jyrkkä, Antti Seppälä, Pete Pietilä
Term and year of completion: Spring 2016, Pages: 38 + 1 appendices

This Batchelor’s thesis was commissioned by Oy LM Ericsson Ab Oulu. The aim of it was to help to investigate and create a test automation solution for the stability testing of the LTE base station.

The main objective was to create a test automation for a predefined test set. This test automation solution had to be created for specific environments and equipment. This work included creating the automation for the test cases and putting them to daily test automation jobs.

The key factor was to create one test set which includes lots of test cases. All of the test cases needed to be automated so there was no need for outside configuration in daily basis. These test cases included steps for setting up the environment, running test sequences and after that reporting test results. The main test automation framework was built on Java and therefore, lots of other functionality was also built with Java

This test automation solution was successfully created and put into daily test automation jobs. The test cases created for the job will continue to be developed further whenever there is a need to add or fix any features. The test reporting part for the test job is reporting all needed information after tests and therefore, this can be used to determinate the state of the software used.

The test automation solution will be used daily for test automation clusters and it will provide lots of information of the current software. This will also free time from the manual testing for other necessary things.

Keywords: LTE, Test automation, Java

PREFACE

I would like to thank Oy LM Ericsson Ab Oulu for providing a possibility to do my thesis work. The idea of the thesis work was customized to match my competences but also to provide useful information for the company. Therefore, my work provided me experience and also something that the company can use. I would like to thank Ericsson's Test Manager Antti Seppälä and Continuous Integration Guardian Pete Pietilä for coming up with this idea.

While working with the topic, I also got a lot of help from other members of the company with various tasks. Especially, I thank Senior Services Engineer Kari Honkasalo for support with LTE specific things and Developer Alberto Gonzalez Escudero for support with various tasks. For the guidance I thank my lecturer Kari Jyrkkä as he always gave me constructive feedback.

I would also like to thank my friend Solution Architect Heikki Laulajainen for all the support he provided while I was doing my thesis.

Now, as I have finished this thesis, I feel that I learnt lots of new things and got lots of good contacts from the business world. I also think that my work was useful for the company.

Oulu, 23.3.2016

Elvis Punnek

CONTENTS

TIIVISTELMÄ 3

ABSTRACT 4

PREFACE 5

CONTENTS 6

VOCABULARY 7

1 INTRODUCTION 9

2 BASIC STRUCTURE OF TEST ENVIRONMENT 10

2.1 General structure of LTE network 10

2.2 Equipment used for the automation 11

2.3 Structure of LTE stability cluster 12

2.4 Future changes of LTE stability test cluster 15

3 TEST AUTOMATION SOLUTION 17

3.1 Test automation needs for LTE stability testing 17

3.2 Parts of automation solution 17

4 IMPLEMENTATION OF THE AUTOMATION SOLUTION 20

4.1 Cluster changes for automation 20

4.2 Configurations 21

4.3 Test place initialization 21

4.4 SW upgrade for the eNodeBs 22

4.5 Traffic sequences for UEs 24

4.6 Fetching and reporting KPIs 26

4.6.1 KPI 26

4.6.2 Fetching the KPIs 27

4.6.3 Test reports 30

4.7 Temporary automation solution 31

5 FUTURE IMPROVEMENTS 33

6 CONCLUSION 35

REFERENCES 37

VOCABULARY

API = Application Programming Interface

BSC = Base Station Controller

CA = Carrier Aggregation

CCN = Cellular Coaxial Network

CLI = Command-line Interface

E-UTRAN = Evolved Universal Mobile Telecommunications System Terrestrial Access Network

eNodeB = Evolved Node B

EPC = Evolved Packet Core

FTP = File Transfer Protocol

GSM = Global System for Mobile Communications

GW = Gateway

JAR = Java Archive

KPI = Key Performance Indicator

LTE = Long-Term Evolution

MME = Moblility Management Entity

RF = Radio Frequency

RNC = Radio Network Controller

SSH = Secure Shell

SUT = System Under Test

SW =	Software
UE =	User Equipment
UI =	User Interface
USB =	Universal Serial Bus
WCDMA =	Wideband Code Division Multiple Access

1 INTRODUCTION

This thesis work was done for OY L M Ericsson AB at Oulu site. Ericsson is an old telecommunication company and it was founded in 1876. Its global headquarters are located in Stockholm, Sweden. It has over 110,000 employees around the globe and in the end of year 2014, 25,700 of them were working in R&D (Research and development) [1]. Oulu site was founded in early 2012 and it got a small team for R&D, mainly focusing on LTE (4G) and WCDMA (3G) radio access products.

The aim of the thesis was to create a test automation solution for the LTE stability testing to test the basic stability cases whenever a new SW (software) package would be available. Before the automation implementation the basic stability tests were already ran, but everything was executed manually. Lots of resources were used for basic testing and automation could free lots of time from testers to manual test execution and fault debugging.

For this thesis there was one daily test chain that needed to be created. This chain included various things such as initializing test place, running stability tests and reporting KPI's (key performance indicator). There was already one in-house automation solution for stability tests, but those tests had no easy way to be started automatically after a new SW package. As there were also other steps in this automation chain, the best way was to create new test steps to complete the chain.

The test automation solution of this thesis was made with an in-house test automation framework built on Java. Test cases were made with this framework and started with a test scheduler in a needed order. The test scheduler also reported all calculated verdicts to a test database and from there it was easy to see whenever a test was passed or failed.

2 BASIC STRUCTURE OF TEST ENVIRONMENT

There are lots of terminology and test environment equipment that are regularly mentioned in this thesis. Therefore, it is important to explain the basic functionality of LTE (Long-Term Evolution) network and the equipment used in this stability test automation solution.

2.1 General structure of LTE network

As the functionality of LTE network is not the most important part of this thesis, this chapter will just describe the very basics of LTE network elements. Below is the image of basic structure of the LTE network.

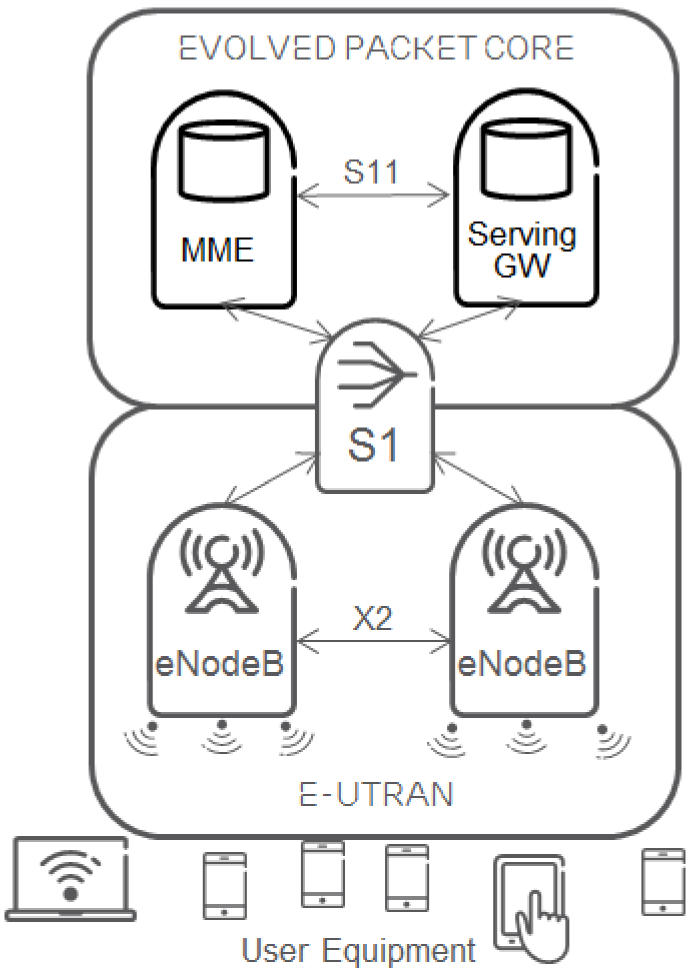


FIGURE 1. Basic structure of LTE network

A general LTE network structure consists of EPC (Evolved Packet Core) and E-UTRAN (Evolved UMTS (Universal Mobile Telecommunications System) Terrestrial Access Network). EPC and E-UTRAN are connected to each other with the S1 interface which supports both user data and signalling. The most important basic components in LTE network are eNodeB (evolved Node B), MME (Mobility Management Entity) and Serving GW (Gateway). [2]

Compared to the earlier radio access technologies such as GSM and WCDMA, in LTE there is not any network controller like RNC (Radio Network Controller) or BSC (Base Station Controller) and the functionality of those network elements is distributed between MME and eNodeB. This makes the network architecture simpler and allows a better performance. In LTE eNodeB takes, for example a resource control and a handover decision from the network controller. And on the other hand MME controls subscription and session data.

2.2 Equipment used for the automation

This part describes the equipment used in the stability test clusters solution which was automated in the frame of this thesis. Hardware-wise these clusters were already built and there was no need to do any major changes. The both automated clusters are similar and have the same kind of equipment. The most important equipment is the SUT (System Under Test). In this case it is an eNodeB radio access product.

For test automation there is a need for many kinds of other equipment. First there is a need for lots of UEs (User Equipment) and control PCs to control these UEs. In this setup there is also a remote controlled power supply for USB (Universal Serial Bus) and power sockets for this equipment. There is also a CCN device (Cellular Coaxial Network) which is an air interface simulator. It creates attenuation between eNodeBs and UEs to simulate a real environment. This equipment will be specified below.

USB dongles are used for this setup to act as UEs which could be used by users in a real environment. These dongles can generate traffic and as there are many of them, they can create a user load. The use of this kind of equipment is important as USB dongles are real UEs and therefore they can create a real load towards eNodeBs. These dongles are connected to controlling PCs and USB power switches.

As there is a need for a user base load, Cobham TM500 is used to simulate multiple UEs. Cobham TM500 is a multi UE simulator and the UEs generated with it can generate traffic as normal UEs. Cobham TM500 also needs its own controlling PC and power switch for its power socket.

All control PCs are virtual machines and therefore do not need as much room as multiple physical computers would need. There is controlling software and needed scripts installed to these PCs and therefore connected equipment can be controlled remotely.

Remote controlled power switches are used to reboot UEs and equipment if there are any problems with them. This way even if the system is bugged or stuck, it can be rebooted and it can retain functionality.

CCN is used to create a simulated network environment for tests. As CCN attenuates signals between different UEs and cells, handovers will occur. With CCN it is possible to simulate moving UEs without moving them in reality. This makes the tests in a laboratory environment simulate the real environment and be more versatile.

2.3 Structure of LTE stability cluster

This part will describe how this test equipment in clusters are connected to each other and what kinds of configuration solutions are used before automation. This part does not describe the test automation solution and only describes how this kind of a setup is used manually.

When testing radio access products, it is important to create a test environment as close to the real network as possible and not to simulate everything. It is also important to choose stable and easily manageable test equipment.

The test clusters used are identical and include:

- 4 eNodeBs
- 1 CCN
- 1 Cobham TM500
- 24 USB dongles (UE)
- 2 servers with 9 virtual PCs
- 2 remote controllable USB power switch
- 1 remote controllable socket power switch

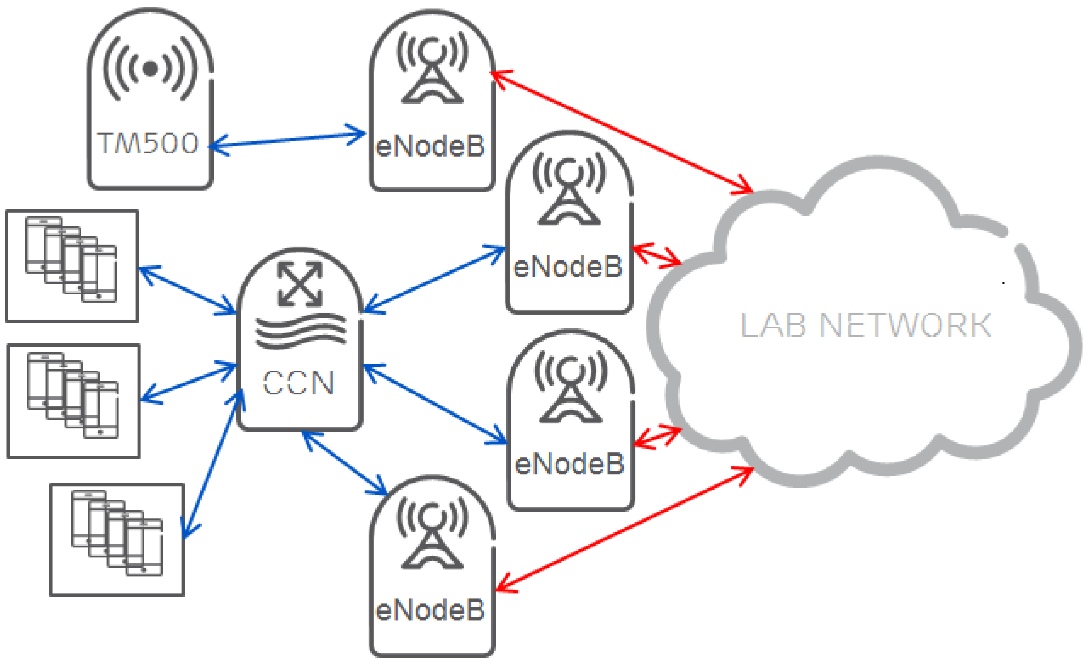


FIGURE 2. Simplified UE connection setup of the stability test cluster

Figure 2 describes how UEs are connected to the lab network via eNodeBs for the stability test cluster. The figure is simplified and does not show, for example all UEs. The USB dongles are located in RF shielded boxes to reduce noise from signals of other eNodeBs.

All UEs are connected with an RF (Radio Frequency) cabling to a CCN. Other end of the CCN is connected to eNodeBs which are connected to the laboratory network. TM500 is connected to its own eNodeB and therefore it is not part of the CCN connection. The laboratory network has servers where UEs can connect via eNodeBs to generate a real traffic. All connections between UEs and eNodeBs are done with a cabling to reduce noise. This kind of setup allows CCN to generate attenuation to separate UEs and eNodeBs.

The laboratory network also includes several servers for eNodeBs to use. These servers include configuration servers where eNodeBs can get its configurations. Also other core elements like MME can be reached via the laboratory network. From the laboratory network testers can connect to eNodeBs to change the settings or get statistics from several different areas. For example, KPI statistics can be received in this way. There is more information about how these KPIs were collected with automation in chapter 3.4.

Other parts of the connections are connections between the test equipment and controlling PCs and power connections of the test equipment. Below there is a simplified picture of these connections.

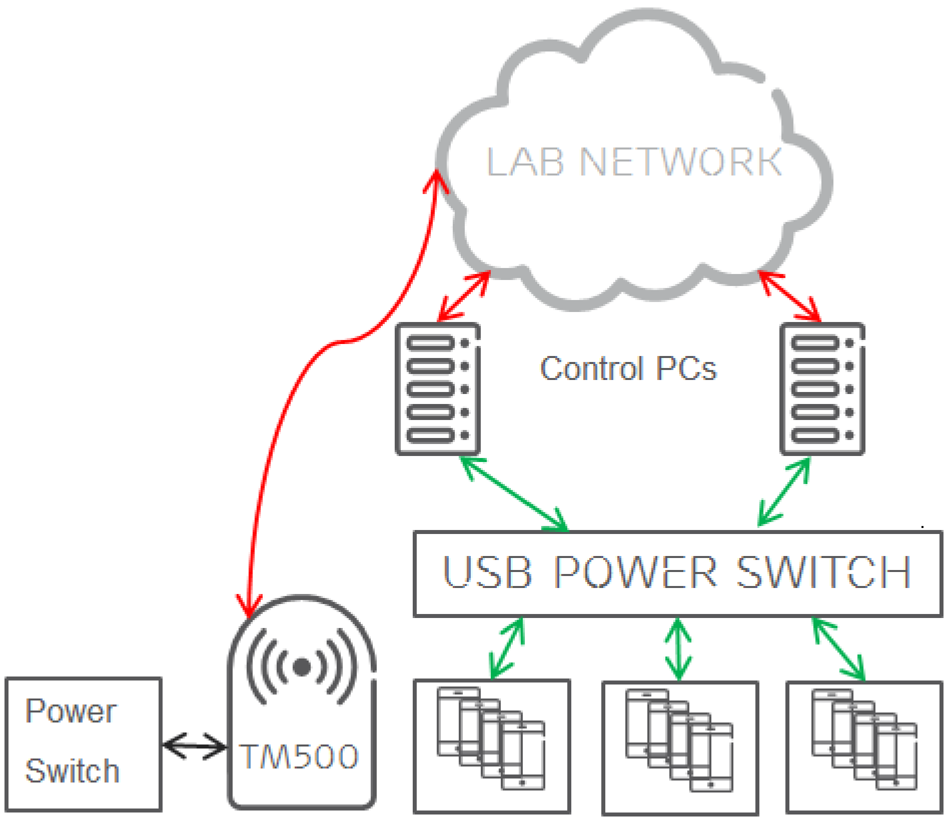


FIGURE 3. Other connections of the test equipment in the cluster

Figure 3 shows how UEs and servers are connected to each other and to the laboratory network. UEs are connected to servers via a USB power switch which can cut this connection between PCs and UEs by disabling the USB port. These control PCs are connected to the laboratory network where they can be reached and controlled.

TM500 is connected to the laboratory network which enables the connection with the TM500 control PC. The main power of a TM500 comes from a remote controlled power switch so it can be easily rebooted if it gets stuck. This is very important for automation but it also makes the manual usage easier.

PCs have their own configuration for testing needs. UE PCs have software which can command the UEs to connect to the servers and do various tasks. This controlling software is also able to collect some information from UEs like throughput measurements and successful tasks. There are also various scripts made for starting test sequences so starting does not always need to be done fully manually.

TM500 controlling PC also got its own controlling software and a FTP server, from where the TM500 can fetch its configuration when started. TM500 also needs its own scripts to make the starting of test sequences easier. Some of these scripts are done with the controlling software of the TM500 and others are done with batch files, which is the own commanding language of the Windows.

2.4 Future changes of LTE stability test cluster

There are incoming changes for the LTE stability test cluster. One of the stability test clusters will be changed to use only 3 eNodeBs. For these eNodeBs there will be 2 cells each. Therefore, connections of the equipment are going to change, but this does not really affect the testing sequences. Figure 4 shows these modifications for the RF connections. As other connections will remain almost the same, there is no need to describe those changes.

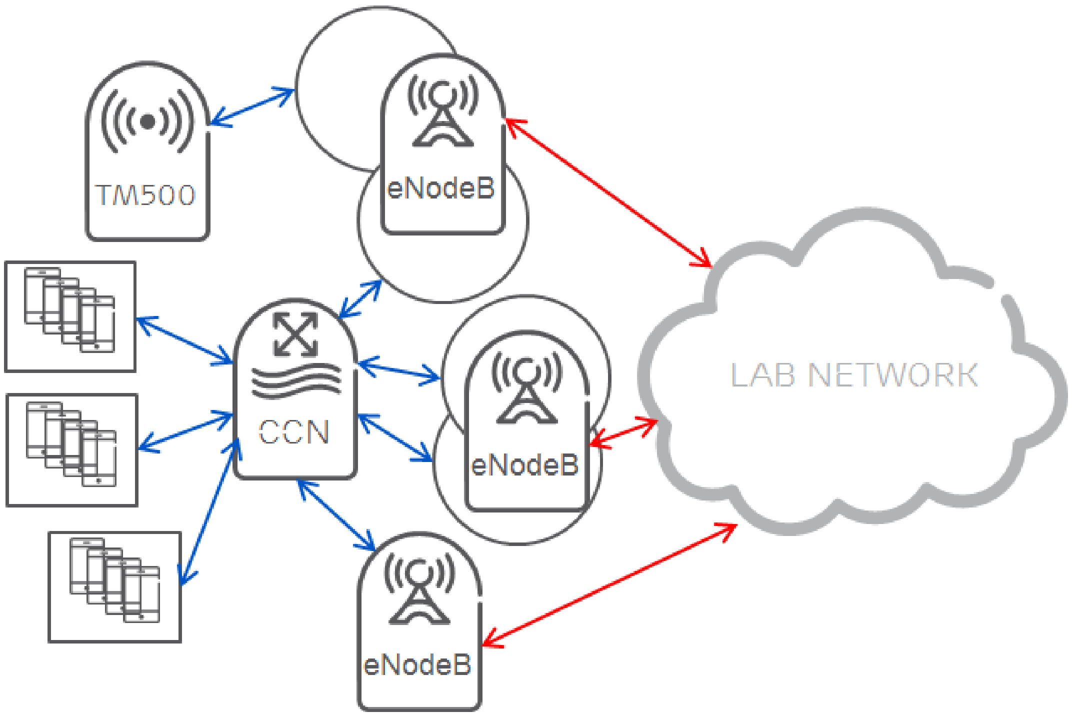


FIGURE 4. Future changes for UE connections of the cluster

In figure 4 circles around the eNodeBs describe cells. The most bottom eNodeB has also two cells but they are used for CA (Carrier Aggregation). When a UE is connected to a CA cell, it can use 2 bands simultaneously to double the throughput. Therefore, there are now 4 different connection points for UEs and one for the TM500. These changes are important because after they have been done there is a possibility to test CA and dual cell properties of the eNodeBs.

3 TEST AUTOMATION SOLUTION

The test automation solution of this thesis consists of multiple parts. This chapter describes what kinds of parts were needed to be automated and why.

3.1 Test automation needs for LTE stability testing

The main reason to automate LTE stability test clusters was to get the earlier results from the newest SW. When new SW packages come at night time and testing is started in the morning, the test results can be received too late to make changes to a new SW. When automating the tests to start after the new SW package at night, test results can be received in the morning and there is more time to make new changes to the SW.

There are also lots of parts in testing, which are frustrating to repeat every day. Automating the whole daily sequence will give more time to test specific things and configurations. There are lots of things that will be tested only once a while and there is no idea to automate them. Well automated daily tests release time to do these tests because then there is no need to fix automation sequences all the time. When the automation is implemented properly, it is able to even fix its own problems with a proper error management.

3.2 Parts of automation solution

The automation solution made in this thesis consists of multiple parts. This chapter will describe these parts. The parts are:

- Initializing the test environment
    - Rebooting PCs
    - Checking that everything is alright
- Upgrading the newest SW to eNodeBs
- Starting traffic
    - Traffic sequences for USB dongles
    - Traffic sequences for TM500
- Fetching the needed statistics from eNodeB and logging them

- Stopping traffic
- Fetching all KPIs and reporting them to the test database

First of all the parts of this automation solution are initializing the test environment which in this case is the whole stability test cluster. The main part reboots all control PCs as controlling functions may get stuck. Rebooting environments is the best way to ensure that the environment will work during of the complete test. This part also includes all other checks that can be done in this part.

The SW upgrade is the next part as it is important that all features will be tested with the newest SW package. This part may not fail as there is not any idea to test the cluster with old software. If this part fails or if there is something wrong with the new SW package, it will fail the whole test automation sequence.

When the test cluster is ready and the newest SW package has been installed, it is time to start the traffic. Traffic will be started at the same time with all the UEs including TM500s virtual UEs. It is important that all UEs are working so the user load is as high as intended. This part was already implemented and only needed functionality for execution and fault management to be added to it. The traffic sequences used by the stability testing are similar all the time and therefore this does not need changes between different executions.

When the traffic load is started for the SUT, there is a need for logging the statistics. There are several different statistics that are good to be logged when the test is running. They can be used for debugging or be monitored while tests are running. These statistics consist of the current functionality of eNodeBs and they will be described further in chapter 4.5.1.

After the intended time has passed, traffic will be stopped from UEs. This part was also already implemented and therefore needed only execution and fault management functionality to be added to it.

The last part of this test automation solution is the test result reporting. This part includes a collection of statistics from eNodeB and calculation KPIs from these reports. There are multiple parameters that will be checked. This one sequence can get results from all of them. For all of these parameters there will be a verdict check to define if the test has been failed or passed. These will be reported to the test database to get long time information of test cases from both of the stability test clusters.

4 IMPLEMENTATION OF THE AUTOMATION SOLUTION

This chapter will describe how all the needed functions were automated and what kinds of solutions were chosen and why. As the in-house test automation framework used in this test automation solution was built on Java, other parts of the automation were also built on Java.

4.1 Cluster changes for automation

When the existing stability test clusters were automated, some changes needed to be done for the configurations of PCs and equipment. The biggest changes were done for the control PCs. While this test automation solution is partly made using the already built in-house test automation solution, there are some configurations and scripts needed to make the clusters equipment compatible.

The most used script for all PCs was a reboot script. This script was added to all PCs to a right folder. Some PCs already had this script, but it was in the wrong folder. Also, when booting the Windows PC, it does not automatically log in. Thus, the programs and scripts in the startup folder do not start automatically. This is not a problem when the PCs are used manually, but when automation is used it is important that PCs log in and therefore start the needed programs. This change was done to the Windows registry. The register changes were done to:

HKEY_LOCAL_MACHINE/SOFTWARE/MICROSOFT/WINDOWS NT/CurrentVersion/Winlogon

Changes done for register:

AutoAdminLogon=1
DefualtUserName=”username for logon”
DefaultPassword=”password for logon”
DefaultDomain=”domain for logon”
delete AutoLogonCount

When the PC would reboot and logon automatically, the programs in the startup folder would start. Therefore, the needed programs and scripts could be added to the startup folder and they would also start automatically after the reboot command. Some of the controlling software could not be added to the startup on their own but needed to be started with the parameters. Therefore, there was a need to create scripts to start these programs with parameters.

4.2 Configurations

All equipment needs parametrized configurations to be run with the test code. These configurations ensure that the same code can be run with different test places. When there is test code that takes configurations as an input, it can easily do same tasks for different test places. These configurations include for example IPs and usernames. There will be more information about these configurations in the chapters below if those configurations are viable to be described.

4.3 Test place initialization

Rebooting controlling PCs was an easy task as there were already rebooting scripts made for the PCs. A reboot script built with a batch includes just one simple command: “shutdown /r /f /t 0”. This command has 3 parameters, which mean:

/r: Not just shutting down but also restarting PC.
/f: Forcing Windows to shut down and not to ask confirmations from the user.
/t 0: Time till shutdown happens.

The reboot script was executed via a Telnet connection to the controlling PC. To use the Telnet connection towards the Windows, there is a need to activate the Telnet server on the Windows. This can be done easily from the Windows control panel by just activating the Telnet from the Windows features. The Telnet service was also configured to start automatically when the Windows would start to ensure that the Telnet service would be always running.

As there were multiple Windows PCs that needed to be rebooted, there was a need to create configurations for every Windows PC. Major configurations included an IP address, a username and a password. There were also some other minor configurations like a connection timeout but they are not so important and could even be left as a default. For the Telnet connection, the in-house framework connection functionality was used and therefore, there is no specification for it.

After connecting to the Windows PCs, just a simple command needed to be sent to the PC to start the restart script. After the execution of the scripts, there was a need to ping the Windows PC to ensure it would reboot and start up again. After rebooting all of the PCs and checking that everything is up and running again, this step could be determined to be successful.

4.4 SW upgrade for the eNodeBs

The SW upgrade for the single eNodeB can take even up to 20 minutes if there happens to be some delay. As this test cluster included multiple eNodeBs, it was wise to create SW upgrades to be running in threads. This would ensure that every eNodeB would be upgrading at the same time and therefore it would save some execution time. As the thread needs to return a value after the execution, there was a need to use the Callable thread [3].

The SW upgrade code was created to start as many threads as there were the configurations for eNodeBs in the test place configuration. This would ensure that the same code could be used for one or multiple eNodeBs and the code would not depend on the amount of the eNodeBs. There are lots of different reasons why the SW upgrade could fail. Therefore, the thread needed to return not only just true or false but also the reason for failing.

Creating a new Java class to be returned from the thread was the best solution to return as many properties as needed. Creating the new class also gives a possibility to add more properties if needed in the future. This class included the Boolean and the String value. The Boolean value would determinate if the SW upgrade would be failing or succeeding. On the other hand, the String value would tell the reason for failing. This String value could also be used to tell other things from the thread, e.g. if the eNodeB would already be using the newest SW.

As there was already a built SW upgrade functionality for the test framework, the contents of the thread did not need to be recreated but the old functionality could be used. As there were some differences between testing the SW upgrade and just upgrading the new SW, so the used SW upgrade functionality needed some adjustments. These adjustments were mainly done to a post check of the SW upgrade functionality.

The SW upgrades thread first checks the version of the newest SW and the SW that eNodeB is using. Therefore, the thread can determine if the SW upgrade is needed or not. If eNodeB is not on the newest, an SW upgrade sequence would start. This sequence includes lots of commands sent to the eNodeB including the information where the newest SW can be downloaded. These commands are sent to eNodeB with the company's internal operations and maintenance CLI (command-line interface), which is not public. Therefore, this part cannot be described more closely.

After the eNodeBs upgrade sequence is ready, the thread does the post check of the upgrade. This includes checking that the newest SW is really installed on the eNodeB. Other important parts are checking that the SW is working. This can be done by checking if eNodeB's cells are up and there are not any functional alarms. After these checks, thread can return the result to the main function.

The main function will wait for threads and after every thread has been finished, it will check the classes returned by the threads. If some of the threads has failed, it will fail the whole SW upgrade case. In any case the last step of the SW upgrade code is printing the results of every node. These results can be used for debugging and it is easy to see which of the eNodeBs has failed its SW upgrade. If all eNodeBs pass all post checks, this part of the automation solution will be determined to be successful.

4.5 Traffic sequences for UEs

This is the first and only step where the traffic controller, which was the already implemented in-house test automation traffic solution for stability, was used. Therefore, this chapter will describe more about using this traffic controller than describing the functionality of this solution. As the development of the traffic controller was delayed, this part was not tested, instead a temporary solution was created and it is described in chapter 3.4.6.

The traffic controller was built on the same test automation framework as the test automation solution of this thesis. As it was built on the Java, it could be packed in an executable JAR file [4]. This made it possible to use a prebuilt JAR of a specific version of this new traffic controller. In this way it was easy to take the specific version into use and build a controlling functionality to work with that version.

As these traffic scenarios made in the traffic controller needed also configuration files, there was a need to execute the JAR with parameters. Multiple execution solutions were considered but at the end the best solution seemed to be a server where all of the needed configurations and the JAR files would be. Therefore, it was easy to build configurations and the test code to execute these traffic scenarios in the traffic controller.

As there were two stability test clusters, it was wise to build new configuration interfaces and not to hard code these cluster configurations in the execution code. These configurations just included names of the configuration files and SSH connection information for connecting to the traffic controller server. Configurations for the traffic controller were created on the server. Therefore, the execution needed only names of the configuration files for executing the traffic scenarios.

As the execution of the traffic controller was a new functionality, it needed its own helper classes for the test automation framework. These helper classes were built to control the traffic controller for different operations. These operations were for example starting or stopping the traffic for all configurations.

As starting and stopping traffic scenarios had separate functions in the traffic controller, it was easy to make own calling functions for them. These functions were built to run in threads so that all of the UEs could be commanded to start traffic scenarios at the same time. This class was easily used from the main function to start or stop the traffic activities while needed. As the needed functionality to control the traffic controller was quite simple, most of the work was done to make as simple but flexible configurations as possible. While configurations were built to be simple, it would be easy to add more UE PCs for traffic scenarios in the future. This kind of setup also makes implementing the future controlling features easier.

As the traffic controller was built on the same test automation framework, its logging was similar to other test solutions built with the same framework. This made it easy to determine from logs if the function of the traffic controller successfully executed or not. In this way, if the traffic for specific UEs was not starting properly, it could be easily determined and the traffic scenario could be tried to start again.

When the traffic had been successfully started, a timestamp from that moment was saved into file for a later use. This time was to be used for fetching the KPIs. The use of this time is specified in chapter 3.4.5.

When the traffic had been started, there was a need to log some specific KPI values to be available through the whole traffic sequence. These values could be read by the testers while tests were running and therefore they would give information if there are already some problems. These KPI values would also tell if there were some bigger problems during the nightly runs for a specific timespan.

These traffic scenarios would be running for a certain time. This time would be over night for every day except for weekends. For weekends the traffic scenarios would last for the whole weekend to get longer stability test runs. This time was easily calculated programmatically in the code from the current timestamp and therefore, it could be determined in the test code.

After this calculated time the traffic scenarios were stopped with the traffic controller. This part works in a similar way as starting the traffic scenarios. As the traffic controller gives logs also from this function, it was easy to determine if the traffic scenarios were successfully stopped.

After all traffic scenarios were stopped, there was again a need to get a timestamp from that moment. This timestamp would also be used for fetching the KPIs. After that, this part of the test automation solution was ready. It would not be so important to shut down traffic or other activities as there was a specific time taken into account when the testing was stopped.

4.6 Fetching and reporting KPIs

Fetching and reporting the KPIs is the most important part of the solution, as it will determine the state of the latest SW. This test automation solution includes various KPIs which give information on different things about tests. As these KPIs conclude if tests are passing, it is important to know what these KPIs consist of. This chapter describes what are the KPIs used in this thesis and how they are collected to the test reports.

4.6.1 KPI

There are 6 main classes of the KPIs. These classes are accessibility, retainability, integrity, mobility, availability and system utilisation. These classes have their own KPI values that are reported. When the KPIs of one class are good, the class can be determined to be functioning. It is important to use KPIs so that reporting of specific things stays similar and testers do not need to invent their own ways to check if something is working.

Accessibility means that the service of the cell can be obtained. This service needs to be obtained within specified tolerances and other given conditions. If accessibility is good, it means that users do not have problems when connecting to the cell.

Retainability means that the service continues to be available. This means that when user is connected to the cell the connection will stay up. Retainability is good if the service is up for a requested duration in given conditions. For the user this means that the connection to the cell will not cut randomly.

Integrity means that the connection to the cell has not any problems. The connection needs to be provided without excessive impairments. This means for the user that the connection can be used as it is supposed to be.

The mobility class includes KPIs for handovers. Handovers need to occur between cells without any problems. When mobility is good, it means that the user can hop from one cell to another cell without any unexpected connection losses.

Availability means simply that the cell is available. This differs from accessibility as it does not compare this connection with the user. If availability is good, the cell is up for the requested percentage of the uptime.

System utilization includes various things such as the hardware usage and traffic capacity. This class means that eNodeB is simply working as intended. It is important that the cell capacity and throughputs are at the intended levels. System utilization also includes management properties and when all these are at the intended levels, system utilization can be concluded to be good.

There are limits for all of the classes and these limits must be reached to be sure that the eNodeB’s SW is good enough. These limits may change due to different equipment and conditions because these may affect the results. If there are some limits with the equipment for some testing, it is usually tested again with the equipment capable to handle the test limits.

4.6.2 Fetching the KPIs

As the eNodeB does not return KPIs from itself, they need to be calculated from various other values generated by eNodeB. This part describes how these KPIs were fetched from the eNodeBs. There will also be information on how verdicts were calculated and parsed from returned values.

As the internal CLI tool is used to gather this KPI information from eNodeBs, it cannot be described in detail. The information on the basic functionality of the used commands can be described but mainly this part consists of the parts implemented in Java.

As the KPIs cannot be fetched straight from the eNodeB, some calculation functionality is built in the CLI tool. These functions calculate some KPIs from the eNodeB but all of the KPIs cannot be fetched with these functions. Therefore, there was a need to create some functionality to calculate these missing KPI values from other values fetched from the eNodeB.

When a CLI command is sent to the eNodeB, it needs some parameters. The most important parameters are a starting and ending time. These timestamps were saved into the file as described in chapter 3.4.4. As the command gets a starting and ending time, it can calculate these KPIs from that timeframe and therefore KPIs can be stated to the testing timeframe. This makes sure that there are not any invalid values from other times when there may not have even been any UEs active.

There are also parameters to choose what kind of view is wanted to be returned from the eNodeB. This ensures that it is an easier way to parse these values from the returned string as the string is always within the same format. When it is known, which KPI value is wanted at a certain time, it can be given as a parameter to print only that value from the list. This also makes parsing the value out of the returning string a little bit easier.

For every KPI report test case, there are the configuration parameters. These parameters include the name of the KPI and the wanted KPI values. As all KPIs cannot be fetched in the same way, there is also a need for some separate test codes to be able to fetch all KPIs.

The returning string consists of some debug print, timeframe and the values wanted. Therefore, the only information needed is the values from the end of the returned string. These values need to be parsed from the returned string. Also, as there may be x number of cells for one eNodeB, this parsed value needs to separate them. The end of the returned string looks like this:

NameOfTheKpiValue   cell1value   cell2value

The values needed are numbers and therefore they need to be parsed from the string to numbers. As the values are in a number format, a verdict check can be done. As the name of the KPI is known, it is easier to build the parser to parse these values from the string.

First, only the line needed will be parsed to one string so that it can be used to parse the values. This was done with the Scanner class of the Java [5]. As the Scanner can scan the string line by line, it is easy to save the line including the KPI values to its own string. This makes it easy to scan KPI values from the single line string.

As the lengths of the values are unknown, it is hard to parse values with known indexes. Therefore the Pattern class was used to parse the values from the string [6]. The Pattern uses a regular expression to parse values from the string. The regular expression can be used in a very complex way and therefore, it can parse various things out of the string. The regular expression used in this case was:

(\d+(\.\d+)?)

As found from the Pattern class' documentation, all symbols have their meanings. For this case, values can have decimal fractions or not. Therefore, there is a need for 2 groups where the second group presents the decimal part if it exists. As the second part has a question mark after the brackets, it means that it may or may not be present. As the second group is in the first group, it includes it, when looking a match for the first group.

When using this kind of regular expression, all the KPI values can be found and stored to the array for a later use. This parsing part is done for all of the eNodeBs that are configured for the test place. Therefore, all of the KPIs from every cell of every eNodeB are stored in an array. These KPIs are converted to a double value to be able to use in calculations.

From the array of the KPIs, the average KPI is calculated. Then the average KPI is compared to the wanted value. If the average is better than the wanted

KPI value, this KPI report test case will be determined as successful. Information about the wanted KPI values is stored into configuration files.

As there are some other KPIs that cannot be fetched in this way and they need to be calculated from multiple different values, there is need for separate test codes to get these values. These test codes work pretty much in the same way but they just fetch multiple values from eNodeB and calculate the needed KPI from these values.

There is also a couple of KPIs that are calculated cumulatively, i.e. they need to be divided by hours of running the traffic. As the time used to fetch the KPIs is not the same as the timeframe, where the KPIs are calculated from, there is a need to parse the timestamps from the KPI return string. These timestamps are not the same, because eNodeB calculates these KPIs only for a couple of times in an hour and therefore these KPI values are from specific timeframes.

These timestamps are parsed from the KPI return strings with the Pattern class as was done with the KPIs. These parsed timestamps can then be transformed into the Date class [7]. The Date class can recognize these timestamps as dates and get the time between them. This time is converted into hours and it can be used to divide the KPI with. Therefore, the KPI can be got for the right timeframe.

4.6.3 Test reports

Test reports are generated from all KPI report test codes. Reporting these KPIs is the only step that provides real stability information and it is therefore the backbone of the test report. It is really important that all of the KPIs are reported correctly. These test database reports only inform if the test case is successful or not. When reporting KPIs, it is also important to keep note of the values. As this system does not support reporting values to the database yet, these values need to be reported separately. As all of the KPI report test codes also log KPIs separately from every cell, testers may find more information on the KPIs from them.

When keeping note of the KPI values, it is important to save the old values to the database. When there is a history of the values in the database, it is easy to keep an eye on the changes of the KPI. In this way, while SW is still at the development phase, developers can see if the SW is getting better or not. These KPIs can also be used to draw a figure to show the evolution of the SW.

4.7 Temporary automation solution

As the development time of the traffic controller was longer than was estimated earlier, a temporary solution needed to be done. Even if the code for controlling the new traffic controller would be done, it could not be used in the automation. The reason for this was the missing functionality of the traffic controller. This chapter describes the temporary solution.

An old traffic controller was used in the earlier test automation solution, but it was not very flexible. However, as this old traffic controller was used for the manual testing, it was decided to make a simple automation to start the traffic for this solution. This solution could be used as long as the development of the new traffic controller would take.

There was not really a simple way to control this old traffic controller as it was meant to be used manually. Its UI (User Interface) was built on the website and the only way to control it was to press buttons on the website. This was hard for the automation as the website might change when updated. The only way to automate the usage of the website was to parse right sections from the html file. As the html syntax stays the same, it can be managed with parsers.

There were some old automation shell scripts built to control this traffic controller but they were not versatile enough. Therefore, there was a need to remake these scripts and build them on Java. Some of the old working parts were called from the traffic controller's server with Java.

The functionality in Java was mainly timing and error management. Old scripts called from the server managed the html files and parsed the needed parts from them. These scripts included programs like wget, gerp and awk. Wget is used to fetch the html page from the website and send commands to the website [8].

Grep can parse the wanted strings from the output [9]. Awk works like a grep but it is also able to do operations to the selected values [10].

To know what to press from the website, there was a need to create template html files which included information on the pressed buttons and checked checkboxes. These templates were prebuilt for both stability test clusters. The parser could fetch the information from the templates and use that information to do things on the traffic controller website.

Even if there is a new traffic controller incoming and some of its features were already implemented, it was important to implement this solution so that the daily automation could be started. After this solution has been created, there will be no need to hurry to use any versions of the new traffic controller as long as the functionality of the new traffic controller will work properly.

5 FUTURE IMPROVEMENTS

As the test automation for stability test clusters may often change and new features will be implemented to them, test cases may need to be updated from time to time. This chapter describes some updates that are predicted for the future needs.

When a working version of the new traffic controller will be released, there will be needs to it that and complete the controlling functionality of its controller. This will be the most important part of the future implementations as the new traffic controller will include some functionality which is not presented in the old one.

When there are new features implemented to the eNodeBs, it may change any part of this automation solution. These changes need up keeping for the test automation solution. And as always, someone will investigate better ways to do things. There may also be changes with the UEs used in the stability test clusters so that the initializing part may need some new functionality depending on what kinds of UEs are added.

Because the traffic controller is all the time getting better and it will get more functionality, there is also need to update the test cases using it. When the planned API has been done, it will be easier to control the traffic controller. Therefore, it may be a good idea to switch the controlling functions of this test automation solution to use traffic controllers API. In this way it makes using the traffic controller more stable and updating the traffic controller will not break any functionality.

The KPI reporting part is made very flexible. Adding new KPI reporting configurations should be easy for new KPIs if needed. These configurations give access to report KPIs that are not yet needed but may be needed in the future. Therefore, the KPI reporting will support new features implemented for eNodeBs.

In addition to known updates, there are always some changes that are not intended. Therefore, sometimes updates in the test automation are needed even if there is not any new functionality. These updates may be very important if the old test cases stop working and there is not any stable reporting happening.

6 CONCLUSION

At the beginning of this thesis work there were lots of unclear parts. There was a need for automation and sequences were pretty much known. But even if the sequences were known, a lot of investigation and planning work had to bee done to find out the best way to implement the automation. While the thesis work continued, it became clearer how all the functionality could be implemented.

As the new traffic controller was chosen to be used over the old one, a lot of investigation needed to be done about the new system. This new system was under development and therefore, there was not really any documentation about how it could be used. This documentation was built at the phase of function development and therefore, only information about the completed parts was available. Some information was also gathered straight from the developers of the traffic controller. At the end all the needed information had been collected and it helped the development of the automation solution of this thesis.

Other challenges with the traffic controller were the incomplete parts that could not be tested. These parts were not finished before this thesis project had to be completed. Therefore, lots of parts were untested and they need to be developed further in the near future. Getting the old traffic controller implemented into the solution also took some extra time as it was not expected at the beginning of the work.

The other parts of the automation solution were much more easily implemented as there was a lot of background functionality that was available to be used. Of course implementing a large amount of parsers took a lot of time. Also, lots of configuration files and test database configurations needed to be done to get all this working.

Even when there were easy parts, there was all the time something changing in the stability test clusters. Due to these changes, some parts needed to be recreated. As the time passed and it was clear that there will be changes, the test cases were built to be more flexible. The most of the flexibility was built into configuration files and therefore the tests do not need to be changed.

At the end there was a complete automation solution that included all planned parts, even if some of it was built on the old traffic controller. This automation solution was taken into daily use and it could save a lot of time in the future. As the tests can be changing in the future and there may be bugs, some up keeping will be needed in the future. But now the testers are very satisfied with the current solution as it saves a lot of time.

In the development process there were lots of new things to learn and investigate. This made the development process of this automation solution interesting. There were also lots of challenges to be solved. As the automation solution was really needed and taken into use, I think this solution was successful. The development will continue and at the end the new traffic controller will be taken into use.

REFERENCES

1. Ericsson. 2015. Facts & Figures. Date of retrieval 18.1.2016. Available: http://www.ericsson.com/thecompany/company_facts/facts_figures

2. LTE Encyclopedia. LTE Network Infrastructure and Elements. Date of retrieval 8.2.2016. Available: https://sites.google.com/site/lteencyclopedia/lte-network-infrastructure-and-elements#TOC-2.-E-UTRAN-and-eNode-Bs

3. Oracle. 2016. Interface Callable. Date of retrieval 17.2.2016. Available: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Callable.html

4. Oracle. 2016. Using JAR Files: The Basics. Date of retrieval 17.2.2016. Available: https://docs.oracle.com/javase/tutorial/deployment/jar/basicsindex.html

5. Oracle. 2016. Class Scanner. Date of retrieval 22.2.2016. Available: https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

6. Oracle. 2016. Class Pattern. Date of retrieval 22.2.2016. Available: https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

7. Oracle. 2016. Class Date. Date of retrieval 22.2.2016. Available: https://docs.oracle.com/javase/7/docs/api/java/sql/Date.html

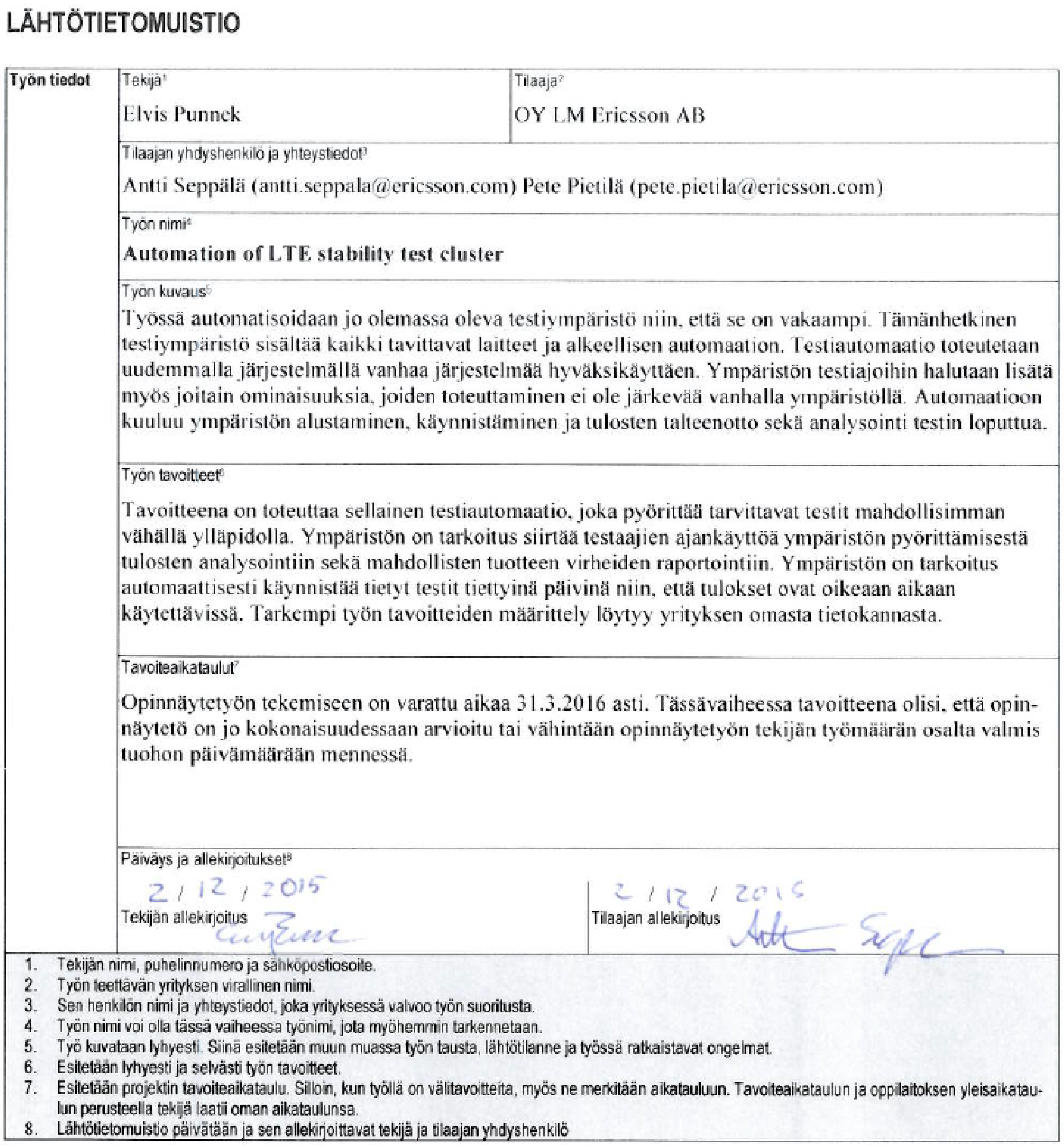
8. GNU Operating System. 2015 GNU Wget. Date of retrieval 22.3.2016. Available: https://www.gnu.org/software/wget/

9. GNU Operating System. 2015 GNU Grep. Date of retrieval 22.3.2016. Available: https://www.gnu.org/software/grep/

10. The GNU Awk User's Guide. 2015 Date of retrieval 22.3.2016. Available: http://www.gnu.org/software/gawk/manual/gawk.html

APPENDICES

Appendix 1 Memo of initial data (in Finnish)



LÄHTÖTIETOMUISTIO

Työn tiedot

Tekijä[1]
Elvis Punnek

Tilaaja[2]
OY LM Ericsson AB

Tilaajan yhdyshenkilö ja yhteystiedot[3]
Antti Seppälä (antti.seppala@ericsson.com) Pete Pietilä (pete.pietila@ericsson.com)

Työn nimi[4]
Automation of LTE stability test cluster

Työn kuvaus[5]
Työssä automatisoidaan jo olemassa oleva testiympäristö niin, että se on vakaampi. Tämänhetkinen testiympäristö sisältää kaikki tavittavat laitteet ja alkeellisen automaation. Testiautomaatio toteutetaan uudemmalla järjestelmällä vanhaa järjestelmää hyväksikäyttäen. Ympäristön testiajoihin halutaan lisätä myös joitain ominaisuuksia, joiden toteuttaminen ei ole järkevää vanhalla ympäristöllä. Automaatioon kuuluu ympäristön alustaminen, käynnistäminen ja tulosten talteenotto sekä analysointi testin loputtua.

Työn tavoitteet[6]
Tavoitteena on toteuttaa sellainen testiautomaatio, joka pyörittää tarvittavat testit mahdollisimman vähällä ylläpidolla. Ympäristön on tarkoitus siirtää testaajien ajankäyttöä ympäristön pyörittämisestä tulosten analysointiin sekä mahdollisten tuotteen virheiden raportointiin. Ympäristön on tarkoitus automaattisesti käynnistää tietyt testit tiettyinä päivinä niin, että tulokset ovat oikeaan aikaan käytettävissä. Tarkempi työn tavoitteiden määrittely löytyy yrityksen omasta tietokannasta.

Tavoiteaikataulut[7]
Opinnäytetyön tekemiseen on varattu aikaa 31.3.2016 asti. Tässävaiheessa tavoitteena olisi, että opinnäytetö on jo kokonaisuudessaan arvioitu tai vähintään opinnäytetyön tekijän työmäärän osalta valmis tuohon päivämäärään mennessä.

Päiväys ja allekirjoitukset[8]

2 / 12 / 2015
Tekijän allekirjoitus [illegible]

2 / 12 / 2015
Tilaajan allekirjoitus [illegible]

1. Tekijän nimi, puhelinnumero ja sähköpostiosoite.
2. Työn teettävän yrityksen virallinen nimi.
3. Sen henkilön nimi ja yhteystiedot, joka yrityksessä valvoo työn suoritusta.
4. Työn nimi voi olla tässä vaiheessa työnimi, jota myöhemmin tarkennetaan.
5. Työ kuvataan lyhyesti. Siinä esitetään muun muassa työn tausta, lähtötilanne ja työssä ratkaistavat ongelmat.
6. Esitetään lyhyesti ja selvästi työn tavoitteet.
7. Esitetään projektin tavoiteaikataulu. Silloin, kun työllä on välitavoitteita, myös ne merkitään aikatauluun. Tavoiteaikataulun ja oppilaitoksen yleisaikataulun perusteella tekijä laatii oman aikataulunsa.
8. Lähtötietomuistio päivätään ja sen allekirjoittavat tekijä ja tilaajan yhdyshenkilö