Timo Räsänen

Improvement of Test Automation

Helsinki Metropolia University of Applied Sciences
Master's Degree
Multimedia Communications
Master's Thesis
20 May 2013

Metropolia

Abstract

Author Title Number of Pages Date	Timo Räsänen Improvement of Test Automation 38 pages 20 May 2013
Degree	Master's degree
Degree Programme	Multimedia Communications
Specialisation option	
Instructors	Ville Jääskeläinen, Head of Programme Tomi Tuhkasaari, R&D Manager

The purpose for this study was to find out how to ensure that the automated testing of MME in the Network Verification will continue smooth and reliable while using the in-house developed test automation framework. The goal of this thesis was to reveal the reasons of the currently challenging situation and to find the key elements to be improved in the MME testing carried by the test automation. Also a reason for the study was to get solutions as to how to change the current procedures and ways to perform the automated testing to fill better the requirements of Network Verification.

The study was carried out by studying the currently used test automation framework and its utilization. That knowledge was then compared to the best practices of the existing test automation methods to find out what should be changed to get the performance and the quality of test automation on the right level. One of the main reasons for this situation is that the fast expansion of the test tool usage and the rapid increase of test coverage with insufficient resources have left the some features, e.g. documentation, lagging behind and now it slows down the debugging and therefore consumes the time resources from further development. One of the proposed solutions for this particular issue is a requirement to add comments on basically every script and code to improve knowledge of the test case to anyone working with it. Some other proposals offer tools on how to decrease the testing time by changing the method of test case creation and also how to modify the existing ones for better performance.

Key words	Test automation, software testing, regression, LASS

Contents

Abbreviations/Acronyms

4G	Fourth generation of mobile networks
eNB	*eNodeB.* 4G base station
EPC	*Evolved Packet Core.* Core network between the RAN (Radio Access Network) and Internet or operator services.
CI	*Continuous Integration.* Software engineering practice where changes are immediately added, tested and reported to provide rapid feedback.
CVS	*Concurrent Versioning System*. Tool used in software development to keep track of changes in a set of files.
F-ISN	*Flexi Intelligent Service Node.* Core network element
ICD	*Intelligent Delivery System.* Discontinued network element
LASS	*Linux Automation Scripting System.* NSN internally developed test automation framework.
LI	*Lawful Interception* is for obtaining data pursuant to lawful authority.
LTE	*Long Term Evolution*. 4G
ME	*Mobile Equipment* equates user equipment, e.g. a mobile phone or a dongle.
MME	*Mobile Management Entity.* Control node for the LTE access network
NeVe	*Network Verification* is the last testing phase in MME testing chain.
NRC	*Nokia Research Center*

NSN	*Nokia Siemens Networks Ltd*
NS	Flexi *Network Server* is the NSN platform of the MME implementation.
Piping	Piping is a method how to export an output of one command to input of the next command.
RAN	*Radio Access Network* is the part of mobile network that takes care of the radio interface to mobile equipments.
SGW	*Serving Gateway* is one network element of evolved packet core.
SUT	*System Under Testing*
TA	*Test Automation*
TE	*Test Environment*
UE	*User Equipment* equates to mobile equipment

1 Introduction

Software testing is a vital part of the process of producing good quality software. Traditionally testing has been done manually, but for the faster development cycles, cost efficiency and repeated testing, nowadays testing is more and more done by test automation. Test automation can be used in all test phases from the module testing to the system verification. Depending on the test phase, the overall coverage of the test automation varies. It is, at least in theory, possible to replace the manual testing with automated testing, but all of it is not worth it. Test automation complements manual testing or the other way round, but in all cases it has to return the investment. There is a number of test automation frameworks available. Some of them are commercial and there are also open source test frameworks available, but sometimes none of them suites some special use. In those cases the whole test framework might be build from the scratch, as in the case this study focuses on.

Nokia Siemens Networks Ltd (NSN) is a telecommunication and data networking company. One product of NSN is the Mobile Management Entity (MME) that is a control node for the Long Term Evolution (LTE) access network. Network Verification (NeVe) is the last test phase in the software testing chain of the MME. Since 2009 NeVe has utilized the Linux Automation Scripting System (LASS) as an automated test framework in automated regression testing. LASS is a completely internally developed test tool. So far, LASS have been proven to be a solid test framework fulfilling the test automation requirements of the NeVe testing. However the continuous development of MME software is all the time increasing the load of test automation. Also the requirements of spreading the test automation coverage have brought NeVe test automation to the point where testing speed and maintenance of test framework is getting worst. There is a risk that continuing this way may jeopardize the test quality and accountability in the future. Nowadays even a whole night is hardly a period long enough to run all the test cases and there are already some issues making usability of LASS compromised.

These expected challenges have brought about the need of studying how the NeVe test automations could be done better. How to make it faster to go through all regres-

sion tests, how to make it easier to maintain and how to keep the quality of testing on a good level or, even better. Thus, the research question of the present study is:

"How to improve Network Verification test automation in MME testing?"

An answer to the question was searched by studying the current best practices of the automated software testing, studying the current implementation of test automation framework and also how it has been used. Also people involved in test automation, from development and management to test engineer, were interviewed to get background information, user and management experiences with visions and also expectations as well as the challenges of the future. This information was analyzed to get an understanding about what has been done right, what needs to be improved and what would be the best way to do it. The final result of this study consists of proposals on what are the main issues to be handled, and what are the most vital actions to take to get those tackled. The proposals should create a guideline for the test automation development to push it onto the right direction on the way to make it a faster, more usable and more reliable a tool.

This study consists of five sections. The first discusses the general knowledge of test automation, the goals, methods, metrics and benefits of it. The second one describes what kind of test automation framework is currently used in the Network Verification MME testing. The third section explains how testing is currently done, how the test automation framework has been utilized and what are the challenges now and in the future. The fourth section is all about the analysis of the current situation on the whole Network Verification MME test automation and pin points the found problems and gives background to the reasons. The last section is about the result of the study. It explains more deeply the found issues and gives proposals how to overcome the challenges found.

2 Software Testing by Test Automation

At the late 70's, it was common that approximately half of time in a software project and more than half of the total costs were spent into software testing. Now, a third of a century later, when new test tools have been developed, new languages created and programmers are used to develop on the fly, the same rule of thumb is still valid. (Myers 2012: ix) Though as the test automation software is getting tested, the work required is not testing, it is coding. Test automation is also a full time effort and cannot be taken as a part time task among other duties. "Test automation must be approached as a full-blown software development effort in its own right. Without this, it is most likely destined to failure in the long term." (Nagle)

Though software testing has a major role in the development and it is the key in getting software working as it should, it still cannot cover everything and there are always some bugs left. It has been estimated that a poor software quality with all the consequences causes annually in the USA alone nearly $60 000 000 000 worth of losses. (Levinson 2011: 3) Though it could be possible, at least in theory, to make software that contains not a single bug, it would require a test system that covers everything, but in that case it would cost so much that it would not make any sense. This is well revealed in the quality dilemma by Bertrand Meyer:

Quality Dilemma

> "If you produce a software system that has terrible quality, you lose because no one will want to buy it. If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway. Either you missed the market window, or you simply exhausted all your resources. So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete." (Pressman 2010: 406)

Test automation can give a good return on investment. However, test automation is not a solution to get all testing done by machines. It cannot replace humans completely. (Koskela 2008: 398) Actually in many cases, test automation coverage of the test cases could be somewhere from 60% up to 85% of test cases. In many cases a human

can see things that test automation has not been coded to check. An experienced test engineer can have an intuition over unusual behavior in a way that machine cannot.

The goal of automated software testing is to prevent defects from slipping into production. (Dustin 2009: 18) When the problem has been found, the test automation can help to reproduce it, even if it occurs randomly and/or rarely. This is one of the benefits of the test automation that tests can be run again and again exactly in the same way while human errors are not making failures to the rest results. The difference between test automation and manual testing is that test automation is coding and therefore it should be done by coders. However, the test automation engineers should not be the same who are doing the coding for the software under testing. If they are, there is always a risk that test cases are done according how the software is coded, and the main reason for testing might blur. Test case should test how the software fills the requirements, not how it has been implemented.

The ideal test automation system guides the test case developer trough the creation process. However, in many cases the test automation platform is providing, so to say, easy language to describe the functions to the test platform to execute. If the test framework is well implemented, it can also do the test result analyzing and reporting automatically. Though, the more complex the test automation system or framework is, the more maintenance it requires.

Coverage

As known, there are basically always bugs in the software. Now, if test automation, or even test engineers do not find errors as much as expected, it could be a sign that the testing is not working as well as it should. But if errors are found more than expected, it could be a sign that the quality of previous test phases, e.g. module testing, is not high enough. (Lehtimäki 2006: 172) Thinking about modern software, they are all for different purposes and functions. Some of them are small, some enormous, but they all have one thing in common. They all have bugs at least on the some stage of the development process. Software testing is not all about finding bad spelling or malfunctioning code, but there are several different purposes for which software testing is

needed. One is to prove to the management that the software fulfills the requirements in both functionality and quality point of view.

Sometimes it might be said that a test run was successful when all cases were passed. Also that testing is set to prove software works, or it shows that software does what it should, but actually its other way around. Testing should add value to product by raising quality or reliability. Therefore a proper definition for software testing could be: "Testing is the process of executing a program with the intent of finding errors." (Myers 2012: 6)

2.1 Testing in Agile Development

When the competition is fierce, customer demands good quality software delivered fast with their requirements, the old software development methods were not fulfilling the requirements well enough. In the early 2000, a group of developers discussed the light weight and rapid development methodologies. They wanted to find the key aspects of the successful development project. The result was a "Manifesto for Agile Software Development". It is the base of nowadays commonly used software development method, Agile.

Manifesto for Agile Software Development:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more." (Cunningham 2001)

Testing in the agile development environment focuses a lot on the customer. Collaboration with customer means that they are involved in the development process as early

as possible and they are also involved to determine what needs to be tested and what are the criteria the software has to fill before they accept it. The main aspect in testing is the customer satisfaction, and when they are involved, it creates confidence to the product and software development and testing. In practice, customers can even define the acceptance tests and the use cases.

"In essence, Agile testing is a form of collaborative testing, in that everyone is involved in the process trough design, implementation, and execution of the test plan." (Myer 2012: 178) When all of this is bundled together, testing is not just a phase, but more like an integrated process in a continuous progress of development. In the agile, the development cycles are short, and even though some manual testing may be required, the automated testing is preferred.

2.2 Test Automation Tools

There are already available quite a few test automation frameworks. Those tools are used as a platform on where to build the specific test automation system. Some of the frameworks are commercially developed, but there are also many open source too, e.g. the Robot Framework and TTCN-3. (Robot Framework 2013)(Testing and Test Control Notation Version 3 2013) Sometimes the available tool does not fit into the special needs of testing. In those cases the only way is to create a new, particular framework from the scratch or to just continue testing manually.

2.3 Benefits of Automated Testing

The only factor of test automation is not that it aims to save the total amount of costs of software developments, but it has many significant benefits too. One is that usually the speed of testing increases dramatically. When manual testing is converted to automation, usually the procedures, e.g. software installation is automated too if not already done. This can speed up testing a lot. And also when the procedures are handled by the scripts, the human delay is vanished. Also the test analyses with returning the test environment back to the starting point decreases the testing time. On the other hand, when the share of automated test cases increases and manual testing decreases, the amount of work does not decrease equally, but the work load moves from

the manual testing to test environment maintenance and basically to the coding work in some form. As explained earlier, the difference between manual and automated testing is that test automation is about coding. More can be achieved with test automation, but in the process to change into it, mainly the tasks changes, not the workload.

2.4 Coverage of Testing

"Coverage is an objective measure that is very useful in software testing. (Fewster 1999: 218) The coverage of testing can be taught from many different angles or it can be a measure in many different levels. One can be a calculation of how many features of all has been tested on some level. Another could be how many messages needed for some feature is tested and third could be that how many of bytes per messages per features are tested. The coverage is used to measure the ROI (Return of Investment) or generally the progression of the test automation project. Though, at least at the start of the test automation project, the coverage seems to be developing linearly toward to the full automation, but in practice, it will never reach it. All tests just are not worth it to automate. Tests having often environmental, or whatever changes, are usually not worth of efforts to automate. Also automating some cases may require some special investments that may be more costly than doing that part if testing manually for a long time.

After the difficulties of a test automation project, progressing into some stage of automation coverage may be quite smooth and relatively easy, but when closing to the last and most difficult test cases, the development of coverage increase will slow down and finally stop to the lack of resources. It is about the quantity and quality of test automation personnel on how fast the goals are reached. It would be a benefit if manual-to-automation conversion is done by testers who have good coding skills. Though, those people should know the system under testing and the manual test cases very well, but after all, the automation work is coding work. Obviously the needed coverage can be reached faster if automated test cases can be inherited from the previous projects or products.

The more the test engineers creating the test automation cases know about the system tested, the more detailed and accurate tests they can create. Another thing is the knowledge of specifications. The deeper the knowledge of specifications and parameters is, the better quality cases can be created. Also the knowledge of the test framework's strengths and weaknesses is a clear benefit for test engineers and therefore to the whole development of testing.

3 Test Automation Framework: LASS

The LASS (Linux Automation Scripting System) is the test automation framework that has been used in the Network Verification tests from the beginning. Test automation from the test network point of view is a parallel testing system with the manual testing, as seen in the Figure 1. Though in this case the LASS is used to test MME in EPC (Evolved Packet Core), any other network element, including 2G and 3G mobile network elements, can be tested no matter what king of mobile network LASS is attached. In this way the LASS is not part of the SUT (System Under Testing).

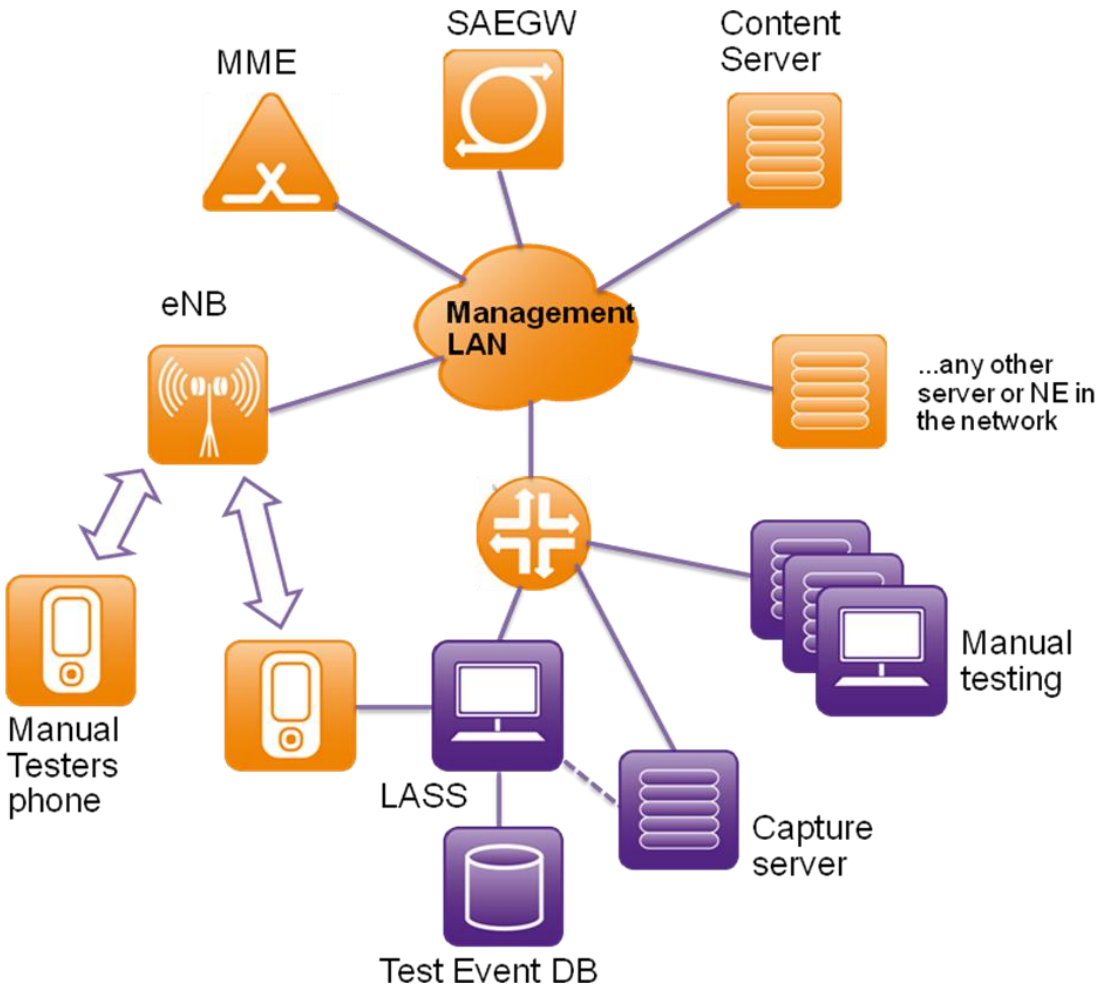


Figure 1. Position of LASS in the test environment

The figure 1 shows the test environment from the LASS point of view and both manual- and LASS testing are not involved in user traffic in any way. In the same way as the

test engineers connects into all of the different network elements and servers trough the management LAN, the LASS also open connections automatically to the same equipments and the test scripts executes the same procedures automatically as the test engineers would do manually. The test engineers are using the real mobile equipments (ME) in the test network and basically in the same way the LASS control the usage of UEs via the USB cable. Instead of the graphical user interface (GUI) of the capture servers that test engineers are using, the LASS script does the same procedures by using the command line interface (CLI).

3.1 Background of LASS

The development of LASS began back on 2001 when all the testing in this part of organization was still manual. There were a need for test automation and already some experiments were made with HIT (Holistic Integration Tester). HIT is also company internally developed and widely used tool within NSN, customers and subcontractor. However, HIT seemed to be too slow to handle testing and analyzes, so another tool was needed and Babic Slaven (Graduate Engineer), who was given a task to get a test automation system, decided to create a new test tool that based on scripts and Linux. Within six months the first version of the LASS engine was ready and the first test cases were successfully run. This approach was much faster and reliable than the HIT-tool. (Babic. 2013)

By the next year 2002, LASS could execute 24 000 attaches and detaches during one weekend without a single error in the test tool. Due the poor results with the real mobile equipments, attach and detach had to be done by emulators instead, but anyway that seemed to be so solid solution that project was continued without real UE's. To being able to control emulators, a little tool was created with four to five commands that could handle the emulators. Test automation in NeVe started somewhere between 2007 and 2008 and the initiator was a need to get tests run with less manual work. During that time also the management got interested of the benefits of the test automation and started to give requests for all test phases to put efforts for getting test automation projects running.

3.2 LASS Language

At the start, before LASS language, there were only those 4-5 commands that were needed to command the NRC (Nokia Research Center) Emulator to get attaches and detached done without real mobile equipment. There was also a requirement from the management to make the test automation system so simple to use that coding skills would not have been required to be able to create automated test cases. Experience from those few commands were so good that the test case coding would be done with those and similar command. That approach seemed to be good enough solution and sort of half way to respond to the requirement of test engineers not needing coding skills. On the LASS version 1.0, there were 17 commands that were enough to create test cases and do the simple analyzing without database. Nowadays there are altogether 25 different commands. There are still no loop- and other commands from the commonly used languages, but anyway those actions can be performed with these. Here is the list of LASS commands.

"COMMAND	USAGE. P1 – P6 are parameters for the command
ABORT	Aborts the test suite and outputs the P1 to stdout.
BACK_POS	Jumps back to line where SAVE_POS P1 was executed
BUFFER	Sets the buffer size in bytes
CALC	Calculates mathematical operation P3 for operands P2 and P4 and stores the result into P1.
CLOSE	Closes the terminal connection
CONT	Outputs P1 to stdout
END	Marks the end of the test case
ERROR	Stops the session with force. Outputs P1 to stdout.
EVENT	Creates an event P1. P2 and P4 can be replaced with P3 and P5.
GET	Waits output for P2 seconds. Stores the string from location P6 (after output is regexp:ed) to P1
GOSUB	Outputs P2 and calls subroutine P1 with arguments P3 P4 P5 P6.
IF	Compares P1 with P3 with operation P2. If true skips number of lines (if P4 is a number) or skips until the label P4
INPUT	After outputting comment P1 expect input from the operator for max P3 secs which fits into regular expression P4 and stores it into variable with name P2
LABEL	Defines the line with label name.
LOGIN	Opens the telnet connection to terminal
PAUSE	Outputs P2 and makes a break of P1 milliseconds
SAVE_POS	Saves a position to be used within BACK_POS
SEND	Sends P1 string to the terminal and cleans the buffer of P2 output
SET	Sets the P1 variable to value P2
SHELL	Sends P1 string to the terminal
SKIP	Skips number of TC's, lines or to the label
SLOGIN	Opens the ssh connection to terminal
START	Logs as event 021 the TC name P1

SUBDIR Changes the TC root directory to P1
WAIT Waits P1 seconds for P2 or P3 to appear as output on terminal"
(Babic 2009: 31)

More detailed usage and command parameters can be found from the NSN internal document: LASS Manual 2.0 (Babic 2009)

3.3 Test Environments

Test automation environments in NeVe are basically similar as manual testing environments. There are not needed any special setups for automation use because all the same equipments are used. Only difference is that there is a PC with LASS installation connected to the test network. When automated test case is ran, LASS open connections to all needed network elements and then it is ready for executing everything coded to the test case. In case of using the real ME (Mobile Equipment), LASS controls it via the USB-cable and simulates the real user. Test environment have also tools to capture traffic from the every interfaces needed. These captures are one of the most important resources when analyzing test cases. Nowadays there are also remotely controllable attenuators that are between the eNB (eNode-B) or base stations and the antennas of those. These attenuators are used to make the air interface to change in a way as mobile equipment would hear the mobile network changing when it is moving. This way mobile equipment is pushed to move from one node to another and when the signaling is captured, all the handover procedures can be confirmed in the analyzes.

3.4 Performance Factors

"How long the test suite takes to run all of the tests is a critical concern. If the test automation takes too long to execute it ceases to add value since the intended feedback is no longer quick (especially for Agile projects with short iterations). In addition, those running the test suite will quickly stop running them since it is just too painful to have to wait that long. Use parallelism, production like infrastructure, or any other trick in the book - but make your tests run fast so that you can maintain quick feedback cycles." (Namta 2012)

The LASS is running on Linux operating system, and it is based on the Korn and Perl scripts along with the MySQL database. That has been proven to be a solid and fast foundation for a test automation platform. However, the system has been expanding since the day one, and nowadays the size of databases is bigger than ever. There are much more analyses, coding for new features, new functions etc. that makes the test case execution slower day after day. It was measured in November 2012 that one empty test case with only the basic pretest and posttest scripts took some 2 min 50 s to execute. On January 2013 the same empty case took 3 min 10 sec, and finally in April 2013, only five months later, the same test took 3 min 45 s. Although when running several test cases in a test run, this time is less significant, it still shows the direction of test time expansion. Those test case initiating and terminating procedures and delays are getting more significant along the increasing number of test cases.

The test case design is one factor that can cause a lot delay to the testing time. Recently there was an incident where the procedure of tested software was changed, and a test engineer created a piece of code that had one minute delay after setting the parameter on. Then he reused the same code to take of the parameter. Later on this procedure was copied to about twenty other test cases causing a cumulated delay of forty minutes to the regression tests. There are also about 51 minutes of delay in the regression test scripts and this figure does not even include any of the delays from all the sub functions similar to the previous example.

Into the performance of test automation system can also be counted in the manual work that the test result handling requires. The ideal test system analyses all of the measures and reports the results without human interference. The contrast of this is a system that gives random alarms of failures though exactly same software is tested by the exactly same test environment with same parameters and settings. These test environment or test framework related incidents create extra work to the test engineers and prevent them from doing beneficial work.

3.5 Software under Testing

The agile software development is generating new software builds rapidly, and whenever these builds pass through the previous testing phases, they finally comes to be

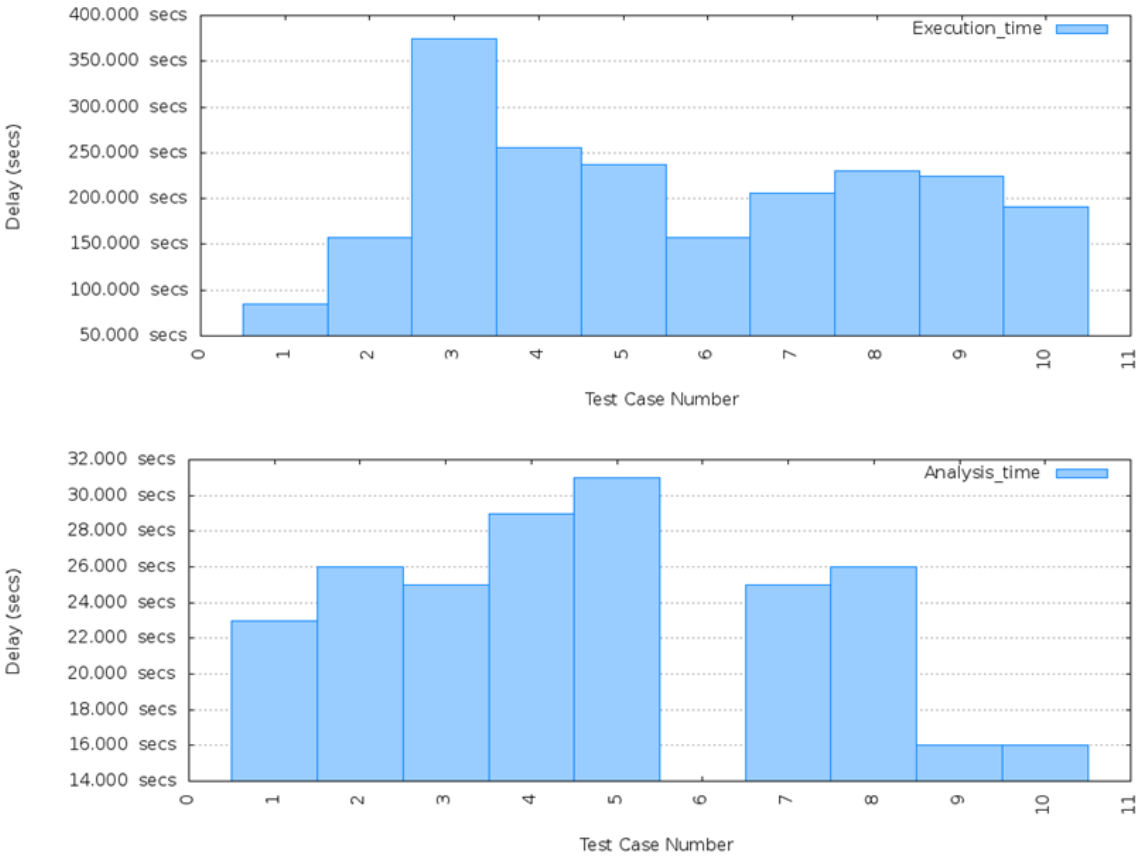
tested in the Network Verification. Some of the test automation environments are set to start automated software installation and commissioning as soon as next suitable time window is found from the test environment booking calendar. When the time is right, the MME is prepared for the software installation by checking the free disk space, some older software package may be removed and then the new software package is transferred, unzipped, activated and commissioned on the spot. There are several different parameters that are used to control of which packages are accepted to be installed on each test environments, so that each environment contains specified packages and debugging with manual testing can be executed between the test automation cases.

3.6 Test Cases and Analyses

Test cases are scripted, or more like marked or described to the LASS by using the LASS language explained earlier in Chapter 3.3. A test case can be simply one single document with LASS commands, but usually they consist of many nested scripts and functions. It is common to make new actions or procedures how to control a new feature to another script that can be reused on the other test cases. Sometimes, when the LASS language is not powerful enough to perform more complicated actions, commands can be given directly to the LASS computer that is running Linux, to get the work done with the help of higher scripting languages. There can be whatever scripts to be run and the results can be exported to LASS variables with ease. One example of this is to use bash script to handle output of a command in test case. That way the exact information of command execution can be put into the variable for later use.

In many cases, the test cases should analyze traffic and check, e.g. is some bit or data correctly sent between network elements. In those cases LASS is set to check that data in analyzing stage and the input for those analyzes are given in the test cases by special events. The analyzing part is done with Perl scripts and consists of both default analyses and events requested in the test case. The time taken by the analyzing stage after the test execution varies a lot according to the needed analyses and how much data there is to analyze.

Figure 2 is from the LASS test reports and it shows the test execution and analyzing times of a test run of a ten test cases.



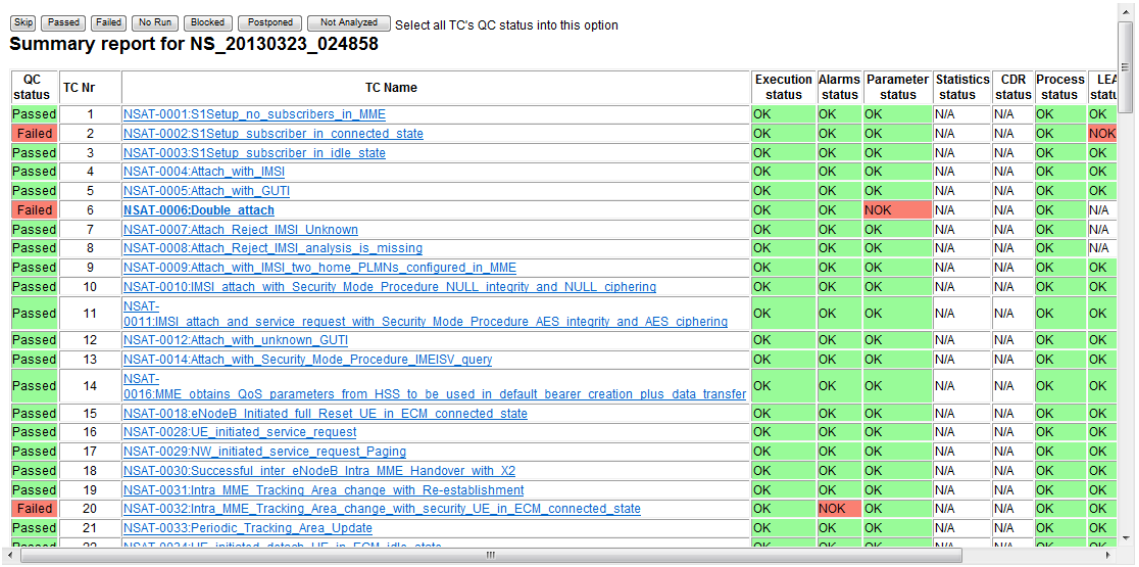
Picture 2 Test execution- and analyzing time

From the picture can be seen e.g. that the test case 6 has failed to execute and therefore analyzing has not been started.

3.7 Test Result Reporting

While the LASS is running the test cases, it saves logs from every opened connection to network elements, user equipments and emulators with execution and analyze logs. Also the traffic from every needed interface is captured and saved for analyzing and debugging purposes. After the test case or test set with many test cases are run and analyzed, LASS creates a local website where all the data is accessible. The database is exported as a web page with alarms, statistics, parameters and other success measures and the test results are reported in web site where the success of testing

can be easily observed. Figure 3, a Summary report, is one example of report pages LASS generates.



Skip | Passed | Failed | No Run | Blocked | Postponed | Not Analyzed | Select all TC's QC status into this option

Summary report for NS_20130323_024858

QC status	TC Nr	TC Name	Execution status	Alarms status	Parameter status	Statistics status	CDR status	Process status	LEA statu
Passed	1	NSAT-0001:S1Setup_no_subscribers_in_MME	OK	OK	OK	N/A	N/A	OK	OK
Failed	2	NSAT-0002:S1Setup_subscriber_in_connected_state	OK	OK	OK	N/A	N/A	OK	NOK
Passed	3	NSAT-0003:S1Setup_subscriber_in_idle_state	OK	OK	OK	N/A	N/A	OK	OK
Passed	4	NSAT-0004:Attach_with_IMSI	OK	OK	OK	N/A	N/A	OK	OK
Passed	5	NSAT-0005:Attach_with_GUTI	OK	OK	OK	N/A	N/A	OK	OK
Failed	6	**NSAT-0006:Double_attach**	OK	OK	NOK	N/A	N/A	OK	N/A
Passed	7	NSAT-0007:Attach_Reject_IMSI_Unknown	OK	OK	OK	N/A	N/A	OK	N/A
Passed	8	NSAT-0008:Attach_Reject_IMSI_analysis_is_missing	OK	OK	OK	N/A	N/A	OK	N/A
Passed	9	NSAT-0009:Attach_with_IMSI_two_home_PLMNs_configured_in_MME	OK	OK	OK	N/A	N/A	OK	OK
Passed	10	NSAT-0010:IMSI_attach_with_Security_Mode_Procedure_NULL_integrity_and_NULL_ciphering	OK	OK	OK	N/A	N/A	OK	OK
Passed	11	NSAT-0011:IMSI_attach_and_service_request_with_Security_Mode_Procedure_AES_integrity_and_AES_ciphering	OK	OK	OK	N/A	N/A	OK	OK
Passed	12	NSAT-0012:Attach_with_unknown_GUTI	OK	OK	OK	N/A	N/A	OK	OK
Passed	13	NSAT-0014:Attach_with_Security_Mode_Procedure_IMEISV_query	OK	OK	OK	N/A	N/A	OK	OK
Passed	14	NSAT-0016:MME_obtains_QoS_parameters_from_HSS_to_be_used_in_default_bearer_creation_plus_data_transfer	OK	OK	OK	N/A	N/A	OK	OK
Passed	15	NSAT-0018:eNodeB_Initiated_full_Reset_UE_in_ECM_connected_state	OK	OK	OK	N/A	N/A	OK	OK
Passed	16	NSAT-0028:UE_initiated_service_request	OK	OK	OK	N/A	N/A	OK	OK
Passed	17	NSAT-0029:NW_initiated_service_request_Paging	OK	OK	OK	N/A	N/A	OK	OK
Passed	18	NSAT-0030:Successful_inter_eNodeB_Intra_MME_Handover_with_X2	OK	OK	OK	N/A	N/A	OK	OK
Passed	19	NSAT-0031:Intra_MME_Tracking_Area_change_with_Re-establishment	OK	OK	OK	N/A	N/A	OK	OK
Failed	20	NSAT-0032:Intra_MME_Tracking_Area_change_with_security_UE_in_ECM_connected_state	OK	NOK	OK	N/A	N/A	OK	OK
Passed	21	NSAT-0033:Periodic_Tracking_Area_Update	OK	OK	OK	N/A	N/A	OK	OK

Picture 3, Summary of test set shows with colors on which area failures are found.

Success rate of LASS or automated regression testing can be categorized to three sections on test case level. Passed-, failed- and Execution Error-cases. The following list describes the determinations when a test case will be treated as a failed:

- Alarms. If during the test case execution any unexpected alarm appeared, or expected alarm did not appear, the test case is failed. Alarms that can exist, but are not necessarily appearing, are usually ignored. In the same way alarms that are expected should be demanded.

- Parameters. On every test cases LASS makes some default checks, e.g. if there are Create-Session-messages, LASS checks are there also Delete-Session-messages. Also there are test case specific checks, called Analyzes, that check some specific messages, bytes, or basically whatever from the captured traffic. These all checks are counted into parameters. If the amount of required parameters doesn't match to found parameters, the test case will be set failed by parameters.

- CRD. Charging Record Data is compared to the user traffic.

- LEA. Lawful Enforcement Agency data from LIB (Legal Interception Browser) is compared to the mobile equipment traffic and interception activation and – deactivation.

- QoS. Quality of Service related data from the traffic is compared to the network element QoS parameters.

The Execution Error is a situation when LASS for some reason failed to execute the test case all the way to the end. This is an abnormal behavior, but on some early test cases it was used also to signal that some critical event has gone so wrong that it is no sense to continue. An example of this could be that some parameter is changed on some network element and because of the command execution fails, the command execution can't be confirmed, or the value of parameter have found not changed after command execution.

A normal procedure after finishing the regression test set is that the results are transferred automatically to the server that keeps tracks of all the reports from all different test environments. From there the test results are easily accessible. The main part of the regression test success is also transferred to other server that keeps track of all NeVe test results. There is also a feature in the local web report where test results can be commented by the test engineers and results with comments, explanations and bug report details can be manually published to those external test report servers.

There is also a couple of other test reporting systems. One is linking the test results to the failure reports and keeps track of the progression of bug fixing and another one is for management and development to follow really fast about the found problems in the tested software or system. LASS can also be set to send e-mails automatically if needed, but a more common way is that the test results are first checked and analyzed by the test engineers and then e-mails are sent if the case requires immediate actions.

3.8 Other Test Tools

The LASS uses a network analyzer to capture data between network elements. The captured data is processed by filtering out exec data and the traffic is exported to the

database. That database is also formed to a human friendly mode by making a table to the report web page. MME contains lots of different computer units with all different logs. For easy access of all the logged data there are a tool called The Log Monster. It connects all those units and collects all the logs into a one directory. There are also many kinds of small tools or scripts that for many purposes that e.g. helps running LASS tests after a period of time.

3.9 Documentation

There is a LASS manual that covers quite well all LASS related information the test engineers might need. There are instructions on how to start working with lass, how LASS language is used, what can be found in the reports and lots of other information. Basically all of this is a list of features named in headers and a brief description or usage of it below. The LASS also contains Linux man-pages. Though there seems to be manuals and information available, it is not that well readable material and the LASS administration manual does not exist. Generally the LASS engine and analyses are not documented at all.

4 Test Automation in Network Verification

The test automation project started in the Network Verification back in 2007 when the system under testing was still SGSN, predecessor of MME. Back then there was one person hired to work partially with test automation. The initiator of the automation project was a need to be able to repeat same tests easily and to reduce the manual labor. Also the management was interested in the benefits the test automation could offer and some kind of pressure was given onto development to this direction.

At the start, there were two competing in-house developed systems for the test automation platform. There were also other test automation platforms available, but not enough resources and competence to evaluate them. One of those two was a HIT (Holistic Integration Tester) based approach, and another one was the LASS (Linux Automation Scripting System). Both of them were somewhat working solutions and the decision between the two was made by the performance and available support of the solution, so LASS was selected to be the only test automation platform for the Network Verification.

The LASS project was already started about six years before and was a fully functioning tool for test automation and it was used already for the ICD (Intelligent Delivery System) and F-ISN (Flexi Intelligent Service Node) testing. According to the management the greatest challenges at the beginning were probably the attitude to change from old manual testing methods to automation. Another challenge was resources, that were insufficient while the old manual testing took so much time that test automation were taken care only as a part time task. After two engineers were hired for 100% automation work, the good results were started came out. So far, the best results has been received from the CI (Continuous Integration) and regression testing where by test automation lots of failures have been found. Though test automation has been giving good results, those could have been reachable from the earlier stage if more resources would have been put and more concentration would have given to the test system stability.

Then also the test automation would have required less manual labor. Also if from the beginning there was own test environments for the test automation, the manual testing wouldn't have been causing troubles to test automation. (Tuhkasaari 2013)

4.1 Regression Tests

In the regression test, there are about 130 test cases testing nearly 700 different measures coded into the test cases, LASS default checks and into analyzes. The number of the test cases is constantly increasing when the rate of automated testing is increasing and also when ever new features are implemented into the SUT. All test cases cannot be converted into automation simply because it is not worth it to give too big an effort to automate something that can be easy to test manually or is often changing. Sometimes new features are not supported by the tools LASS is using. For example, some protocol might not be all the way supported by the network analyzer, or there are not yet mobile equipment emulators or mobile equipments available that supports some new feature that is already in testing phase on test networks.

4.2 Automating New Feature Testing

Among testing fixes of the existing software, one main factor is to test new features. Those are continuously developed and added into the software, so also the test automation needs to be updated. At the beginning, the features are manually tested in the similar test environment where the test automation case will be created. Then the test case is coded and after the results are verified manually, there are input for automated analyze to be implemented onto test framework. Finally, when the feature is working solid, test case runs without problems and analyzes found required changes, the new test case is added to the regression test set.

The initiator of creating a new test case can come from many different inputs. Usually when development adds new features to the software and the process is still ongoing, the specifications of the feature are also available for testing, so the test case also can be created. Another major initiator is reported problems found from the software. These can be found and reported by the own- or other test phases, or even from the customer. Many times the customers are also involved in determining what are needed to

be tested from the software or feature they have requested. In some worst cases the customers have even found issues that have to be fixed and tested in the future.

4.3 Creation of Test Cases and Analyses

When a new test case is created, the input of that may comes from the several different sources. At the start of test automation, the main procedure was to transform the manual cases to the automated test cases. Since that work is quite a lot done already, nowadays the input comes mainly from the new features. Whenever the new feature is in the stage where it can be tested, it will be investigates to found how much of it can be tested automatically and how much have to be done manually. Also the test environments might need some adjustments or configuration to get the feature supported. New test cases are done also when a bug have been found, reported and fixed. Then the root cause of the failure is explored and if old test cases cannot be adjusted to make them find the failure, then a new test case is created.

In some cases it is not needed to rebuild the old test cases to make them detect some other failures. If the flow of the test case suites, it might be just enough to create just a new analyze for the existing test case. Analyzes differ from the test cases in a way that they are not done in the test case, but in the LASS code with Perl scripts instead of LASS language. Analyzes are run after the test case is executed, so the captured traffic is available to be examined. The input of adding a new analyze comes usually from the test engineers who have studied a new feature and planned the test case.

4.4 Reporting

LASS creates test reports automatically whenever test cases are run. However, test engineers are needed to analyze the reports to see if reported failures have been fixed, whether there are new ones or if there is something wrong in the test environment. No matter if the found problems seem to be already reported, or if they are environment failures that are not caused by the software that has been tested, the found error has to be analyzed every time. In the case of a new failure, a test engineer creates a failure report to the failure tracking system where it is given to the responsible team to be fixed. The environment failures are not necessarily reported anywhere, but fixed by the

test engineer or by the test laboratory support team. After the test engineer has analyzed the test run, it is commented and reported further as described in Chapter 3.7

4.5 Test Framework Maintenance

Whenever the LASS need tweaking, whoever is able to make the necessary changes, is allowed to do it. There is no control of who does what. This is a fast and efficient way to make the LASS to adapt into new requirements. However, since this is done in many campuses by many people, for serving testing of different products, every now and then it happens that someone makes a change that seems to be working for him/her, but breaks something elsewhere. Those can be tricky and require time consuming efforts to find the original problem from people who have no idea why suddenly something does not work. And lack of administration documentation hits the hardest in these cases.

5 Analysis of Current Stage

Test Automation in MME NeVe testing seems to be rather sustainable. Test coverage is increasing and the chain from automated software installation and regression testing works like a charm. However, there are weakness and shortages that need to be treated.

5.1 Test Coverage and Automation Level

Since the automation of SGSN/MME in Network Verification was effectively started for some five years ago, the automation coverage has been increased up to 85% of all test cases. But as the matter of fact, the number of all testable items as only somewhere near 45 %. During this time the total number of test cases has increased smoothly up to 130 test cases. Currently NeVe is aiming to increase the number of test items up to 62 % which makes roughly 830 test items. However, the previous percentage of test coverage does not seem to give an accurate picture about how much has been tested with test automation. When a test case is manually tested, the test engineer with his/her experiences can recognize abnormal activity, but automated testing detect only what it has been set to check. Therefore the total coverage of automated testing may be superficial. Many times, in the currently automated test cases, the main part of, e.g. the features are checked, or from the signaling only the vital bytes from one message is inspected.

There are probably many reasons for this, but one that could be changed is how the goals and qualification of successful automation are set. Just the mindset of how progression of automation is calculated. So far, it has been calculated on how many test cases are created, and this might have led, with or without a calculated decision, to create just easy test cases. Instead of that kind of an approach, the measures about how test automation is done should include items that create an aspiration to the implemented test cases in a way that they give the maximum benefit for the software development.

One thing that seems to be left out earlier is that the test automation should be a full time effort. Since in the past test engineers have been pushed in many teams to start doing test automation along with their manual testing, it has not shown good results. People have been frustrated to get poor result with lack of competence and they have preferred to continue testing with the old manual methods. The other thing related to this is the fact that the test automation work is coding, not testing. People, who are involved in the process of test case creation or modification need coding skills and a coding mindset, regardless of the LASS language that is a lot easier to learn than more sophisticated programming or scripting languages.

5.2 Quality of Test Automation

Quality can be taught from many different aspects, but here the focus is on the quality as to how accurate and reliable the information is what the automated regression testing outputs by default, the word *poor* describes it well. Currently the situation is that many times less than in half of the test cases in regression testing are completely passed. There are lots of alarms coming from the tested system and some analyses show wrong results because they have been originally set checking slightly different things and there are also failures in the test framework that e.g. alarms for the missing traffic, though everything is found from the traces. Many of these are already well known issues and e.g. many of alarms have nothing to do with test case or tested items. This gives false results and causes extra work to the test engineers who are analyzing the test results. The reason for this situation seems to be clearly a lack of resources that have caused a rat race. People who are working with the test automation do not have time to fix the problems in the test environment, so those problems keeps coming up again and again and it slows down the test result analyzing. At the same time, new test cases should be created, old ones adjusted to work correctly in the constantly changing environment.

To cut a long story short, the main factors jeopardizing test automation quality seem to be:

- Same test environments are tried to be used for test case creation, testing and test environment maintenance at the same time and all of these operations are bothering or preventing each others.
- Vulnerability in both the LASS and the test case coding because of no visibility of where some change is possibly causing failures.
- Test environment network and network element outages and wrong configurations.
- Problems in software commissioning due SUT not configured for changes or due other commissioning issues.
- Problems in other network elements, e.g. changed configuration in SGW (Serving Gateway) or not fully working LIB (Legal Interception Browser).
- Faulty code that works locally, is committed to the CVS (Concurrent Versioning System), but changes cause test automation to fail elsewhere.

In addition, the future features should be studied and the preparation work should be done so that testing could get started as soon as new features are ready to be tested.

5.3 Documentation

There is a lot to be done for the documentation. It seems like when things have started from the small, there has not been a request for documentation, but instead information has been shared easily by talking with the team mates. However, now when test automation has a significant role in the whole testing and the use and maintenance of the LASS test framework has been spreading, it is clear that the documentation is lagging far behind.

LASS Documentation

The LASS user documentation has been done for some four years ago and just some things has been updated afterwards. The LASS manual for users seems to be covering all of the needed information. However, the outlook of the documentation is rather

dismal, and looks like it has been written without thinking about the reader who does not know much about LASS. One shortage in LASS documentation is the administration manual, or actually the fact that it does not exist. Currently the LASS engine, so to call, is a bundle of interworking scripts and there not a single flow chart of how testing or analyses are done or explanation what is done in which scripts and how they are tight together. The only way to track failures is to read logs and scripts backwards to find the original cause of the failure. This situation for sure is getting better after getting some experience about how things are implemented, but at the same time the code is continuously edited in many places and sometimes changes make functions unrecognized again.

Test Case Documentation

The input for test cases comes usually from an already documented failure, feature or other request and therefore the source of test is available. But when the test case is joined to the test set, e.g. to the regression tests, the trace to the source of test case is cut out. Currently the MME regression set has about 130 test cases and the testing team and is getting more people working with it. That is why there should be a clear procedure about how to verify the problems found in the test case. Sometimes, even though the LASS language is quite easy to read even by th beginners, it is needed to have information easily available about what the test cases are doing, what the special requirements are and what goes wrong if the test case is abnormally aborted. There is neither a list of test cases that clearly shows what they test.

Analyze Documentation

Since the analyzes are reusable checking tools, it is important to have documents about what analyzes already exist, what are their requirements when taken in use, what they test and what parameters are required. Currently there is just a long script where all different analyzes are coded with or without comments about the usage in the test cases. One lack of analyzes are also that they does not report the failures clear enough to the test reports or logs. Originally analyzes has been made for looking the all different things from the captured traffic, but nowadays they are used also for data comparison, calculations etc.

Testing Time

As mentioned in the setup of the study, there are testing time issues to be solved. Currently the regression takes already too long time and the situation is getting worst. There are some clear subjects, e.g. delays in the test cases that should be removed and the amount of test cases are too much for the current setup of the test framework. Besides of those, LASS is getting slower by the increased lines of code to be executed and the size of databases.

Time is not wasted only in the test execution, but also in the manual test result analyzing, because of the false alarms that poorly coded test cases, analyzes and unstable test environment with all the network elements are causing. Testing time is also consumed by handling of the captured data. The quantity of captured traffic is so huge that it is difficult or even impossible to make test cases more than 15 minutes long. Currently there should be a test case to test counters that have a minimum of 15 minute periods before the counter can be verified, so it is under investigation how these tests can be implemented. There are some methods to tackle this, e.g. reducing idle signaling by smarted traffic filters, but the work is still ongoing. The test environment has also a problem with the multiplied packets in the captures that increase the size of captures.

6 Recommendations

There are several things in the MME Network Verification test automation that work very well, but as a result of this study, this chapter explains the main findings and gives recommendations related to them.

6.1 Reduction of Regression Test Execution Time

By the time when the automation project was started and the manual test cases were started to be transformed to the automation, the measure of that achievement was how many test cases were automated. Though one test case usually includes several things to be checked before a test case could be passed, the same way of calculating was also used when talking about e.g. rate of test automation or progression of the transforming work. This seems to have led into the situation where a test case was a measure for success of test automation development. Back then, when there was a hand full of automated test cases, measuring this way was fine from the test framework point of view. Test runs ran fast through, no matter on how many test cases e.g. one feature was divided. However, nowadays when there are over hundred of test cases, the execution time is measured in hours, and is closing to the limit when one night is not long time enough to run the regression set.

Arranging Test Cases

The more there are test cases and the more features are tested, the more complex automated testing and analyzing is required. All the time there are coming more code to be executed on every test case, because more code is added to the common part test execution. During the period of three month (12/2012 – 02/2013) the execution time of the simplest attach – detach test case have increased from 3 minutes up to 3 minutes and 20 seconds. This increased twenty seconds cumulates in 130 test cases up to 43 minutes of increased testing time. From this short test case, it has been estimated, that nearly three minutes of total test time per test case is the general part of test case execution and analyzing time. That part makes about half of the whole testing time on regression testing.

Solution for the test cases arrangement would be:

- New test cases should be added to the existing ones if possible.
- Short and simply test cases should be joined together.
- Test cases of each features could be in a one single test case
- Compressing the size of databases with a more dynamic table creation instead of the same static table on each test case.

Now, how these are put in practice. The first and second one can be done by any test automation engineer, but the feature related more complex cases should be joined together by the one who has create it. The dynamic data base is a trickier task and requires consulting of a MySQL and Perl expert.

Reducing Static Delays

Another time saving action would be to change static pauses from the test cases to dynamic state checking loops. At the moment there are about 52 minutes of pauses only in the regression test run codes, and there are also lots of pauses in the sub functions that are not counted in to those 52 minutes. The number of pauses in milliseconds from the regression test set NSAT.tc have been printed out with the following piped grep and perl commands and can be used later on to follow-up the changes:

```
grep -i pause NSAT.tc |grep -vE [#]|grep -vE [\"].*[\"]|grep -Eo [0-9]+00 | perl -lpe '$c+=$_}{$_=$c'
```

That command filters out all lines containing case insensitive word "pause" from the NSAT.tc file and pipes those to another filter that clears the comments out from the result lines. Those lines are filtered again to remove anything else but series of numbers that have at least two zeros after numbers. Finally those numbers are calculated together with perl.

As an example, the System_Restart -test case have a static five minute pause for waiting system to recover from the restart. This and other delays, that are made long enough to cover the worst case scenario, should be replaced with a wait loop. Since

the system restart takes nearly two minutes, the time saving on each regression set for this particular case would be about three minutes. This might sound invalid factor, but those three minutes cumulates annually into ten hours of testing time per actively used test environment.

6.2 Quality Improvements

Quality is the factor that testing should protect. Currently there are some issues weakening the test quality and therefore the quality of the product is in jeopardy. Some of the quality issues are about the testing quality and some of them are the quality of the test environment and test tools. Also one quality matter is the coverage of tests that basically determines how much faults can pass through the tests, all the way to the customers.

Test Environments

Currently the NeVe test automation is suffering from too many test environment faults. By the definition of the management, the rate of passed test cases per test run should be at least 95%, but currently if not counting in the big number of test environment faults, it is somewhere between 80 and 90 percent. Many of failed test cases have already reported problems that are going to be fixed according to the class of the fault, but the majority of the failed cases are, either test environment failures, or originated from the testing system itself. Nowadays those environment failures just come and go, and those are too often ignored as they are not caused by the software under testing. The first step to tackle these problems is to keep record of them and their repetition. A simple Excel-sheet with rows for failures and columns for software version and date would work. Another step should be a zero tolerance for environment errors. Since those are causing extra work and taking time from other activities, fixing of them should be prioritized high.

Test Coverage of the Test Automation

Some of the test cases have been created about three years ago when the knowledge of the features was just became available and test cases were created. Now those fea-

tures are more familiar to the test automation personnel and when the knowledge has been increased, it is suggested to take advantage of that knowledge and make more coverage to the test automation. More captured messages can be checked and deeper analyses can be created. How to overcome this is that basically all of the test automation cases should be divided to the people who have the best knowledge to get them checked again to found more details to be tested. Currently there are also new analyzes that might be able to be reused in the older cases whenever similar activity is found. Currently only a fragment of all the messages and values are tested. Since all cannot be checked, the shotgun test strategy could be applied. (Black 2007: 40) In practice it means that if it is not known what should be tested, the parameters, values, bytes and messages are randomly selected from the mass to be tested.

6.3 Testing of Fault Correction

Sometimes there is an urgent need to get NeVe tests done as soon as possible to get the software package delivered to the customer. However, if both emulated and real equipment test sets are run, the NeVe MME regression tests take over ten hours. This delays possible further fault management significantly. For these cases, there should be a predefined smaller test set that could be use at the beginning of the regression testing. These sets should be focusing into defined areas, e.g. to charging, and would most likely to be the ones that will reveal the existing problems from the software concerning that area of testing. These test sets should be easy enough to select so that whoever responsible of running regression can be sure what is the most needed test set. This way the focused testing need can be executed sooner, the main results can be gone through faster and the problems can be reported while the other regression tests are still running.

Tough there is a possibility to run tests for several software levels, currently there are named test environments for each level and the whole testing per software level is done by one automated test environment. On the fault correction testing, the other environments can execute simultaneously other of these test sets, while the focused test environment is running the target set. Now, if this divided test set method was taken one step further, the other test environments, when those are not having their own testing ongoing, would run part of the small test sets whenever needed. For sure,

this would require some special efforts to get all of the test environments to share testing, but would be worth further study.

6.4 Test Platform

The problems in testing that are caused by the test platform, originate mainly from three different sources. One of them is the test framework maintainers, who are doing modifications to keep their own testing systems up to date. These changes sometimes cause malfunctioning on other test branches. Currently there are not practices how to make sure changes works for everywhere, or even how to inform everyone needed about the changes. This leads to the situation where malfunction in test framework causes extra debugging work and delays reporting of test results. The solution for this is to imply normal software development methods to the test framework development. This would cut the current method of committing an individual files to the content management system, but instead collecting all changes into the test platform software update packages that are tested in relevant test environments before delivered to the production use. And even though this solution would not be taken in use at once, the temporary solution to filter out the problems caused by the faulty code is to take a local backup from the LASS before updating it from the CVS. Then the test results of both LASS versions can be easily compared.

Another test framework related problem is the always changing system under testing. There are protocols changing, new features developed that are not yet supported by the test tools or test environments. These changes could be anticipated by issuing a responsibility for someone to give input from the development as soon as coming test requirements are known so that coming changes could be planned an implemented before the due date.

6.5 Documentation

It is a must to have a proper documentation process just like in everywhere else, also in the software testing. Currently there is not a LASS administration manual, so basically the only documentation concerning the maintenance of LASS, are the notes and comments in the LASS scripts. This has an influence on a quite small number of peo-

ple, but depending on the coding skills and debugging competence, it may be a major slow-down-factor for the problem solving process. Also the slow debugging obviously delays the software testing and therefore delays the whole chain of software development and delivery.

Also the test cases are lacking documentation. Basically all of the test cases are written to LASS bases on the manual testing test-descriptions, but those are not linked to the test automation test cases, except by the name that usually describes what the test case is all about. The third thing is the analyses that are easily reusable, but detailed description of actions and requirement for the analyses are missing. This leads to the situation where test engineers cannot see why an analyze fails to find something when the searched data is clearly visible to the test engineer. Reusing the same analyses on other test cases might seem to be working properly, but in some circumstances it fails. Also modifying old analyze to a new situation may break the original use of it and the rat race of fixing that analyze is ready. All of the above mentioned cause extra work in failure debugging

Proposed actions:

- Test case scripts should have an outline description of what is tested and how results are analyzed. Also the pronto numbers and the original test case description details should be added.
- Analyzes needs a better comments and logging to help maintenance and debugging of found problems.
- When analyze sets the test case parameter to fail, it should export exact information to test log about what is checked and how the same data can be evaluated manually.
- Test cases that are made according the failure report should have a direct link into failure reporting system so that it's fast and easy to check what the test case should exactly do.

All of these proposals are aiming to increase the performance of fault management. However, there is also the lack administration documentation. This is one thing that should be sorted out with the people involved to LASS development to find out what

kind of documentation would be most beneficial and not only be done for the sake of getting documentation.

7 Discussion and Conclusions

During this study many areas of the test framework, utilization, working methods and test automation practices were analyzed and a lot have been found to be adequate and coming up to the expectations. However, there is a demand for getting the reliability and performance of the test automaton into a sustainable level, where test results are reliable and the further development of the test environments, test framework and test cases is secured.

Thinking about the improvements of the Network Verification MME testing, many things are already in a good shape. LASS language is an excellent approach to get the test engineers without proper coding skills to get rather easily familiarized with the creation of automated test cases. Also the LASS engine is fast and reliable. In spite of all solid parts of the test automation, many things have been found to require improvement. Probably the most vital action is to get all of the test environment related alarms, parameter failures and quality issues solved that are loading test result analyzing and currently preventing further development of the test automation. Another resource saving action for the future is to get the documentation of the test cases and automated analyzing on level that even the new comers can debug failures that are found by the test automation and report them further with a confidence about what has caused the occurrence.

After having that in order, the next in line is to increase the quality of test cases. It means that all of the test cases should be checked from weakness, pauses should be transformed to check loops, all relevant analyzes should be added on every test cases and maybe even simple test cases should joined together. The purpose of this is to avoid slipping again back to the same situation where test automation currently is. After the MME NeVe testing has been put back on trail, all of the other improvements mentioned in Chapter 6 can be verified, planned and implemented.

Because of the history of test automation in the Network Verification MME testing and the background of the used test framework was slightly known already from the be-

ginning, the outcome of this study was somewhat expected. At the start of this study there was a clue that there must be e.g. unnecessary pauses and documentation was probably not up to date, but so many other minor things came up along the way that all were not even covered here. With a clear test automation strategy and needed resources and competencies the weaknesses found maybe could have been avoided. Hopefully the proposed actions give enough power for the current tools to be able to handle the test requirements for a long time. However, since the LASS seems to be in the maintenance mode, or at least not actively developed further, some day the other test automation platforms may need to be evaluated to find out what tool would best answer the new challenges of the constantly evolving requirements of test automation.

References

1. Babic, S. 2009, LASS 2.0 manual - Version 1.5, Espoo: NSN internal document.

2. Babic, S. LASS Developer, Interview, Apr 2 2013

3. Black, R., 2007.Pragmatic Software Testing, Becoming and Effective and Efficient Test Processional. Indianapolis: Willey Publishing, Inc.

4. Cunningham, W., 2001, Manifesto for Agile Software Development, [WWW document] www.agilemanifesto.org (Accessed Apr 19, 2013)

5. Dustin, E., Garrett, T., Gauf, B., 2009, Implementing Automated Software Testing – How to Save Time and Lower Costs While Raising Quality. New York: Addison Wesley

6. Fewster & Graham, 1999, Software Test Automation, New York: Addison Wesley

7. Koskela, L., 2008. Test Driven, Practical TDD and Acceptance TDD for Java Developers. Greenwich: Manning Publications Co

8. Lehtimäki, T., 2006. Ohjelmistoprojektit Käytännössä, Ohjeita Ohjelmistoprojektien Johtamiseen. Jyväskylä: Gummerus Kirjapaino Oy

9. Levinson, J., 2011. Software Testing with Visual Studio 2010. New York: Addison Wesley

10. Muers, G., Badgett, T., Sandler, C., 2012. The Art of Software Testing - Third edition. New Jersey: John Wiley & Sons, Inc.

11. Nagle, C., Test Automation Frameworks, [WWW document] http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm (Accessed Apr 2 2013)

12. Namta, R., 2012, Thoughts on test automation in agile. [WWW document] http://www.infoq.com/articles/thoughts-on-test-automation-in-agile, (Accessed Jan 3, 2013)

13. Pressman, R., 2010. Software Engineering: A Practioner's Approach, Seventh Edition. New York: The McGraw-Hill Companies, Inc.

14. Robot Framework – Generic test automation framework for acceptance testing and ATDD, [WWW document] www.robotframework.org (Accessed Mar 31, 2013)

15. Taylor, R., Medvidovic, N. Dashofy, E., 2010. Software Architecture Foundations, Theory, and Practice. New Jersey: John Wiley & Sons, Inc.

16. Testing and Test Control Notation Version 3, [WWW document] www.ttcn-3.org (Accessed Mar 31, 2013)

17. Tuhkasaari, T., R&D Manager, Interview, Apr 4 2013