Abstract

In Ericsson, the Automation Team automates test cases that are frequently rerun. This process involves copying data related to a particular Configured Test Case from a database and then pasting it into a java file created to run a test case. In one java file, there can be more than one Configured Test Cases. So information can vary. Then the tester has to add package name, necessary imports, member variables, preamble and post amble methods, help methods and main execution methods. A lot of time and effort are consumed in writing the whole code. The Automation Team came up with a proposal of having a tool that can generate this whole information and the tester just has to add or remove minor changes. This will save time and resources. So the development of tool started and finally a tool named Automatic Test Builder developed in java was created to help automation teams in Ottawa, Kista and Linkoping.

This document elaborates problem statement, opted approach, tools used in development process, a detailed overview of all development stages of Automatic Test Builder. This document also explains issues what came during the development, evaluation and usability analysis of Automatic Test Builder.
Acknowledgements

Thanks to all mighty Allah, creator of this universe. He gave me strength to start, continue and complete this major minestrone of my educational career. I am also thankful to my parents, my wife and my siblings to aid me with their prayers and moral support. I am also humbly thankful to my supervisor Kristian Sandahl who led me throughout this thesis work with his valuable guidance and succored me with his valuable experience. I also thanks to my supervisors at Ericsson Jonas Widén and Sören Andersson for their protagonist vision resulting in a quality product. I am extremely thankful to all my friends and well-wishers for their encouragement and adherence.

I dedicate this thesis work to my family.
# Table of Contents

Abstract ................................................................................................................... 3  
Acknowledgements ...................................................................................................... 4  
Table of Contents ......................................................................................................... 5  
1 Introduction ............................................................................................................. 1  
   1.1 Problem statement ............................................................................................. 1  
   1.2 Context of study ............................................................................................... 1  
   1.3 Teams interaction in Ericsson .......................................................................... 2  
   1.4 Approach .......................................................................................................... 3  
       1.4.1 Development method .................................................................................. 3  
       1.4.2 Verification and validation ......................................................................... 5  
       1.4.3 Tools .......................................................................................................... 5  
       1.4.4 Programming Language .......................................................................... 5  
2 Contribution ........................................................................................................... 5  
3 Theoretical framework ............................................................................................ 6  
   3.1 Software testing ................................................................................................. 6  
   3.2 Testing levels .................................................................................................... 7  
       3.2.1 Unit Testing ............................................................................................... 7  
       3.2.2 Pros and Cons ........................................................................................... 7  
       3.2.3 Stubs and Drivers ..................................................................................... 7  
       3.2.4 Integration Testing .................................................................................... 7  
       3.2.5 System Testing ......................................................................................... 11  
       3.2.6 Acceptance testing ................................................................................... 11  
   3.3 Testing modes .................................................................................................... 12  
       3.3.1 Black-box Testing ..................................................................................... 12  
       3.3.2 Pros and Cons .......................................................................................... 12  
       3.3.3 White-box Testing .................................................................................... 13  
       3.3.4 Pros and Cons .......................................................................................... 13  
       3.3.5 Grey box testing ....................................................................................... 14  
       3.3.6 Pros and Cons .......................................................................................... 14  
   3.4 Testing Types ..................................................................................................... 14  
       3.4.1 Functional testing ..................................................................................... 14  
       3.4.2 Pros and Cons .......................................................................................... 14  
       3.4.3 Non-functional testing .............................................................................. 15  
       3.4.4 Regression testing .................................................................................... 16  
       3.4.5 Progression testing ................................................................................... 16  
       3.4.6 Automation testing .................................................................................... 16  
       3.4.7 Pros and Cons .......................................................................................... 17  
   3.5 Hardware and tools ........................................................................................... 17  
   3.6 JCAT framework ............................................................................................... 18  
       3.6.1 JCAT Layers .............................................................................................. 18  
       3.6.2 JUnit 3 ....................................................................................................... 19  
       3.6.3 Subversion ................................................................................................. 19  
       3.6.4 Eforge ........................................................................................................ 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6.5</td>
<td>Maven</td>
<td>20</td>
</tr>
<tr>
<td>3.6.6</td>
<td>Hudson</td>
<td>21</td>
</tr>
<tr>
<td>3.6.7</td>
<td>Sonar</td>
<td>22</td>
</tr>
<tr>
<td>3.7</td>
<td>Information model</td>
<td>23</td>
</tr>
<tr>
<td>3.7.1</td>
<td>FV Legacy team</td>
<td>24</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Automation team</td>
<td>24</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Work Package</td>
<td>24</td>
</tr>
<tr>
<td>3.7.4</td>
<td>TC suite</td>
<td>24</td>
</tr>
<tr>
<td>3.7.5</td>
<td>TC</td>
<td>24</td>
</tr>
<tr>
<td>3.7.6</td>
<td>CTC</td>
<td>24</td>
</tr>
<tr>
<td>3.7.7</td>
<td>TC Header</td>
<td>24</td>
</tr>
<tr>
<td>3.7.8</td>
<td>Preamble</td>
<td>24</td>
</tr>
<tr>
<td>3.7.9</td>
<td>Post amble</td>
<td>25</td>
</tr>
<tr>
<td>3.7.10</td>
<td>Help methods</td>
<td>25</td>
</tr>
<tr>
<td>3.7.11</td>
<td>Main execution TC methods</td>
<td>25</td>
</tr>
<tr>
<td>3.7.12</td>
<td>Test methods</td>
<td>25</td>
</tr>
<tr>
<td>3.7.13</td>
<td>Signum</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Implementation</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Overall structure of the application</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Screen shots of application</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Structure of java file</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Class diagram of java file</td>
<td>40</td>
</tr>
<tr>
<td>4.5</td>
<td>Inputs and output of the application</td>
<td>41</td>
</tr>
<tr>
<td>4.6</td>
<td>Manually created java file VS Java file created by Automatic Test Builder</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>Development process</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Semi-automated process</td>
<td>42</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Generation of text file</td>
<td>42</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Reading text file and saving data in local database</td>
<td>43</td>
</tr>
<tr>
<td>5.1.3</td>
<td>TC header generation</td>
<td>43</td>
</tr>
<tr>
<td>5.2</td>
<td>Fully-automated process</td>
<td>43</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Connection to database using SOAP service</td>
<td>43</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Java file generation</td>
<td>44</td>
</tr>
<tr>
<td>5.3</td>
<td>GUI Design</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>Working GUI</td>
<td>44</td>
</tr>
<tr>
<td>5.5</td>
<td>Adding main execution TC methods and help methods</td>
<td>44</td>
</tr>
<tr>
<td>5.6</td>
<td>Creation of help methods class hierarchy</td>
<td>45</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Creation of Hardware methods class hierarchy</td>
<td>45</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Creation of System Function Group methods class hierarchy</td>
<td>46</td>
</tr>
<tr>
<td>5.7</td>
<td>Documentation development</td>
<td>47</td>
</tr>
<tr>
<td>5.8</td>
<td>Training of testers</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>What they did before</td>
<td>48</td>
</tr>
<tr>
<td>6.2</td>
<td>What they can do now</td>
<td>48</td>
</tr>
<tr>
<td>6.3</td>
<td>Testing of the system</td>
<td>48</td>
</tr>
<tr>
<td>6.4</td>
<td>Estimations of experts about the application</td>
<td>48</td>
</tr>
<tr>
<td>6.5</td>
<td>Future Improvements</td>
<td>49</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Problem statement

At the Long Term Evolution Radio Access Network (LTE RAN) Integration & Verification department, verification and troubleshooting of new and legacy features is performed. The test cases (TC) that will be frequently re-run are automated. Automation of the test cases is carried out by the Automation team using JCAT framework. All support for JCAT is provided by Test Automation Core (TAC) Team. According to the Team Lead of automation team Jonas Widén “It takes about two weeks to write an automated test case”. Because they have to first write all information about each Configured Test Case (CTC) included into a test case. There can be one or more than one CTCs in a single test case which means that huge information is needed to be written or copy and paste from the database each time a tester automates a test case. Then they have to follow a standardized structure to write an automated test case. They have Help methods which they have to write from scratch or copy and paste from the central repository each time they create an automated test case.

All above mentioned steps take around two weeks or more to be completed. If the tester is not skilled in automation, it can take even more. To minimize the time spend on adding CTC information, creating a structure of the automated test case and adding methods, Team Lead automation Jonas Widén and Senior member automation Sören Andersson came up with an idea of having a tool that can do all tedious jobs so that they can speed up the automation process.

The work consists of development of a tool that allows a tester to enter CTC ID and all information regarding that CTC ID shall be fetched from database. This information shall be editable before putting this information into the TC header as comments. Every CTC belongs to a specific System Function Group. This tool shall add all help methods of that System Function Group to the java file. There shall be an option to select hardware. Based on hardware selection, help methods regarding hardware shall be added in the resulting java file. Finally, the tool shall ask for the test case name and path where java file will be saved. After providing all the information, a java file with TC header, complete test case structure, Main execution methods and help methods shall be created.

The result of the work shall be presented to the testers in department. Instructions on how to use and configure the tool shall be made available on internal wiki pages.

Development of such a tool that can speed up the automation process was requested by automation team. This tool will be used by automation team. So, client and end user of this tool is automation team.

1.2 Context of study

This thesis work was carried out in Ericsson Linköping. The department was Feature Verification. The Team which was initiator or this thesis work and which will use the end product of this thesis work was Automation team. There are three automation teams, in Linköping, Kista and Ottawa. The automation team in Linkoping consists of three testers and a Team Lead. The automation team automates test cases that are frequently rerun. Test cases are given to the automation team for automation by Feature Verification (FV Legacy team). So, it can be said that the client of the automation team is Legacy team. But other teams can also run the automated test cases. The automation team uses JCAT as testing framework. All the test cases are developed in Java language and are run in JCAT environment using eclipse IDE. All support related to JCAT is provided by the TAC team.
In November 2011, the automation team was created. They started working on their first automation project in January 2012. This project contains eighty-five test cases that will be automated by the automation team. To speed up their automation process, they proposed a tool that can generate a structure of test case and tester can fill in missing code. Finally development of such tool now named as Automatic Test Builder was started in February 2012. As described in section 1.4.1, prototyping was selected as development method. The whole development process was divided into several implementation phases. After completing each implementation phase, a prototype of the Automatic Test Builder was presented to the automation teams in Ottawa, Kista and Linkoping. Their feedback was taken and the development of new prototype was use to start. This process was continued until all the requirements were implemented. Testing was done throughout the development process. Usability testing was done by automation teams.

1.3 Teams interaction in Ericsson

Figure 1, Teams interaction model

Figure 1 shows team interaction model. There are three automation teams, In Ottawa, Kista and Linkoping. Client of Automation teams are FV legacy teams. Automation teams communicate with FV Legacy teams to get test cases. These test cases are written by FV legacy teams. Then the automation team automates the test cases and delivers automated test cases. Other teams can also use those automated test cases.

All kinds of JCAT support is provided by TAC team. If automation team needs JCAT support classes, they will ask TAC team.
1.4 **Approach**

Thesis work started by reading necessary documents and studying current work flow of automation team so that a better understanding of their technical terminologies and a good sketch of how they work can be developed.

Work started by having several meetings with automation team so that they can express and discuss their requirements and a better idea of what automation team really wants can be attained. After meetings, development of very first version of the product was started.

1.4.1 **Development method**

Prototyping was used as the development method to interact with the automation team. The main reason behind choosing prototyping was to involve the automation team in all development stages and in all decisions about how the end product should be. By doing this, validation of the product is carried out frequently. Figure 2 shows different prototypes of the tool. This figure also shows major features implemented in each Prototype. In each new prototype, improved features were improved from the previous prototype. For example as shown in following figure, in prototype 5, GUI was improved from the GUI of prototype 4. Detailed description of each prototype is given in section 0.
Figure 2, Prototypes
1.4.2 Verification and validation
Verification and validation is the process of testing and inspecting the software to ensure whether the software is according to the customer’s expectations. Verification is an internal ongoing process to ensure that we are developing the system right. Whereas, validation is done by the end of each phase to ensure that we are developing the right system. For validation, customer checks the system and gives a verdict that this product is according my needs or not.

For verification, continuous testing was carried out throughout the development process by me. Validation of each prototype was done when the prototype was shown to the automation team. After they have validated each prototype, development of next prototype was started. This ensured high quality of the product.

1.4.3 Tools
Eclipse IDE was used for the development of ATB (Automatic Test Builder). There were two main reasons for selecting Eclipse as development environment. First was that, Automation team which will use ACG is working in Eclipse to run their test cases. So, there will be fewer problems for the client in the maintenance of Automatic Test Builder. Secondly, in the initial requirements, client expressed the desire to have an Eclipse based application.

1.4.4 Programming Language
Java was used as programming language because Automation team is working in java. Java is an open source language with widely available support. So maintenance of the product becomes very easy.

2 Contribution
This report contains a brief description of whole thesis work. It explains different implementation phases. This report educates the reader by discussing different testing techniques specially software test automation. After reading this report, the reader will have an idea of how to plan thesis work and how automation is carried out in Ericsson. This report emphasizes the importance of client involvement in development process. If client is involved in all development phases by giving his feedback then we can minimize the risk of implementing changes late in project which are extremely difficult to handle. Design of any project is a base for development. If a change in requirement demands design change then it will consume huge time, effort and money to implement that change. So by reading this report, a reader can understand the importance of strong client involvement. Client involvement was the main reason for choosing prototyping as development method. For more information about development method, please see section 1.4.1. Objective of automation team behind this thesis work is to have a tool that can save their time and efforts consumed in gathering required data from different places and putting it into a single java file. They want to standardize structure of test case so that all test cases should have same structure. This will make it easy to maintain test cases. They also want to remove the variations in versions of help methods. Summing it up, automation team wants that this tool should generate a java file containing a structure described in Figure 29.
3 Theoretical framework

3.1 Software testing

There are many definitions explaining testing but according to IEEE Standard 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology", Testing is "The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.” Above definition incorporates the whole testing process. Testing constitutes running a system in controlled environment and observing behavior of system and depending on behavior, verdict is given about the system. [9]

Lee Copeland in his book “A Practitioner's Guide to Software Test Design” described testing as “testing is the process of comparing "what is" with "what ought to be.” [3]

“comparing What is with what ought to be” refers to comparing actual results with expected result.

Software testing is a process in which the system under test is analyzed whether is it doing what customer wants from it. The aim of software testing is to help designers in making a product that is capable of performing desired operations. Testing techniques are selected depending on which software aspect is of more importance for the customer. For example for web sites, customer wants that his website should run on all web browsers. So for this, compatibility testing will be performed. If the requirement of the client is to perform compatibility testing and the testing team performs usability testing then the end product will probably be something which client does not want. So, choosing right testing technique depending on the requirement of the customer is extremely vital in developing a high quality product.

Following are some of the most important testing techniques. Among bellow described testing techniques, some were used in the thesis work depending on the customer’s requirements. For more details about different testing techniques, please refer to reference. [21]
3.2 Testing levels

3.2.1 Unit Testing
In unit testing, a smallest testable code from the whole system is selected, it is segregated from the system and its behavior is analyzed. Each unit is tested before integrating them to form a complete system. Units can be imagined as blocks, when they are tested and combined, a complete system is formed. For unit testing, Stubs and Drivers are needed to be written. If top down approach is used then stubs are written and if bottom up approach is used then drivers are written.

If a system has for example two units and these units are not tested before integrating them together then problems can arise in anyone of the two units and finding the root cause of that problem will be difficult because tester has to look into the whole system. On the other side, if each unit is separately tested then any bug in unit one will be isolated and fix without taking care of unit two and same will be the case for unit two. So unit testing allows isolating bugs and fixing them separately. Finally, after all units are independently tested, they can be combined to form a complete system. Integration testing will be done of that whole system. For more information about unit testing, please refer to reference. [13]

3.2.2 Pros and Cons
Unit testing enables a tester to find such classes or methods that are not behaving according to the specifications, providing the information about correctness of code. As aim of unit testing is to find bugs that are lurking on low level so this testing approach does not consider the whole system or how these units will communicate with other units, leaving defects that can occur when different units intercommunicate. So it can be said that unit testing does not test the design of a system. Testing a small unit of a big system is very simple as compared to testing how different units are working together to achieve the final task.

3.2.3 Stubs and Drivers
The stubs are usually written by testers and are dummy units that act like real units. They only return the value which calling unit needs. Logic is not implemented in the stubs. The stubs are used in Top down integration approach. Whereas the drivers have less throwaway code as compared to stubs [12] and are used in Bottom up approach.

3.2.4 Integration Testing
Integration testing can be said as testing of interfaces of independently tested units. Integration testing is performed after unit testing but before validation testing. For validation testing, please see section 1.4.2. When unit testing is successfully performed and all units are ready to be integrated, units are combined together and their interfaces are tested. The aim behind integration testing is to know that units are interacting with each other properly to complete a collective task. Inputs are given to the integrated system and outputs are analyzed. All units should collaborate with each other as they are intended to do and the system should generate the expected result. There are four ways of integrating a system.
3.2.4.1 Top down
First one is top down, in top down approach, integration starts from upper level and goes to the lowest level.

Let, there is a system containing seven units. These seven units are independently tested. Now, they have to be integrated as shown in Figure 3. In top down approach, unit A will be integrated with stubs of units B and C. So, unit A will be tested by integrating it with the stubs of unit B and C. When unit A will be completely tested, stubs will be gradually replaced by real units. This process will be followed from top that is unit A to the bottom that is unit D, E, F, and G. As bottom most units are leaves, so no stubs for them are required. For top down integration, nodes-1 stubs are required.

3.2.4.2 Pros and Cons
Test cases are written keeping in mind the functional requirements of the system under test. So, the defects in design of the system are uncovered earlier in the testing process. In top down integration approach, no drivers are needed. But there are also some draw backs of this approach. As in top down approach, high level problems are uncovered but there is a great probability of leaving technical details which results in uncovered low level defects. One other limitation is that if the unit for which stub is to be written is very complex that is having a lot of conditions than writing stub for that unit will be very hard.
### 3.2.4.3 Bottom up

Second is bottom up approach, in this approach, integration starts from bottom and goes to the top most level. In this approach, drivers are used instead of upper level units. If a system shown in Figure 4 is under bottom up integration, then first, units D and E are integrated with the driver of unit B. After testing units D and E integration will move to next level. This will continue until top most unit is integrated. If top down and bottom up integration approaches are compared then it can be said that less drivers are needed than stubs. As in Figure 3, six stubs are needed to test the whole system. But only three drivers are needed to test the same system using bottom up approach. For bottom up integration approach, nodes – leaves drivers are required.

![Figure 4, Bottom up](image)

### 3.2.4.4 Pros and Cons

When bottom up development is used then using bottom up integration testing approach is more worthy. Low level details are focused more which results in uncovering more defects on low level. But on the other hand, low level components are usually available off-the-shelf. Usually bottom up integration testing approach is very useful when there is a system with real time requirements. But limitation of bottom up integration testing approach is that as testing is started from low level so the user feedback about the system is postponed which can result in developing a system which user probably has not asked for. In bottom up integration, drivers are written which are more complex and harder to write than stubs. [16]

### 3.2.4.5 Big bang

Third approach is Big bang, in this approach, all units are integrated at once and then whole system is tested. Integrating the whole system at once saves time but also introduces the difficulty in fault isolation. If a system has many units then tracking fault becomes extremely difficult.

### 3.2.4.6 Pros and Cons

Big bang integration testing approach is perhaps useful for small systems. The units and their interfaces should be well-defined to be tested using big bang integration testing approach. As the whole system is integration at once so no stubs or drivers are needed but this also introduces a problem of fault isolation which makes it very hard to find whether the bug is in a unit or in the interface of the unit. This also incorporates the chance of skipping extremely important bugs that should be uncovered during testing of the system. Integrating the whole system at once also makes it difficult to confirm test case coverage.
3.2.4.7 Sandwich
The last approach is sandwich approach. This is a combination of top down and bottom up approaches. In Sandwich approach, a middle level is identified. Top down approach is used from top most level to the middle level and bottom up approach is used from bottom level to middle level. In some cases, a sub tree is integrated and tested using big bang and other sub trees are integrated using top down or bottom up integration approaches. By using this approach, less number of stubs and drivers are needed to test the whole system. For example, in Figure 5, two stubs and two drivers are needed. But fault isolation is compromised. [17]

![Figure 5, Sandwich integration](image)

3.2.4.8 Pros and Cons
In sandwich integration testing approach, the whole system is tested in a gradual manner. If the system crashes, newly integrated component is analyzed. The testing progress can be easily verified against the decomposition tree. The limitation of sandwich integration testing approach is that it is assumed that the structure and units are correct. So testing can only be performed on correct structures. As sandwich integration testing approach is a combination of both top down and bottom up integration testing approaches, so both stubs and drivers are required to be written. If any change occurs in any unit, the whole system has to be retested. [17]
3.2.4.9 Pair wise integration
Pair wise integration is performed using a call-graph instead of using decomposition tree as used in top down, bottom up, big bang and sandwich integration testing techniques. The main benefit of pair wise integration is that no stubs or drivers are needed in this type of integration. Real units are used instead of investing efforts in developing stubs are drivers. In pair wise integration, one integration session is used to integrate one pair of units. In Figure 6, six sessions are used to test whole system.

![Figure 6, Pair wise integration](image)

3.2.4.10 Pros and Cons
There is increase in number of sessions but extra effort is saved that is consumed in writing stubs and drivers. The drawback of pair wise integration testing is that if a bug appears in a unit, let’s say unit B in Figure 6, it can be seen that unit B is used in three different pair wise integrations. Bug will be fixed but those three pair wise integrations will have to be repeated and retested.

3.2.5 System Testing
System testing is performed on a complete system after it is integrated. System testing is carried out after integration testing. The tester does not have to have knowledge of internal structure of the system under test. Tester will give inputs and analyze outputs. The aim behind system testing is to check whether the whole system is producing right results. The System should implement all specified functional requirements. The whole system is considered as a single unit. System testing includes, functional testing and performance testing. Functional testing validates functional requirements whereas performance testing validates nonfunctional requirements. For more details about system testing, please refer to reference [15]and reference [12]

3.2.6 Acceptance testing
Acceptance testing is done to validate the requirements. It involves end user evaluation about the end product. There are special tests that are designed for acceptance testing. These special tests are called benchmark tests. Bench mark tests are test cases that are executed on different products from the same category to have comparison of new product and its competitors. Pilot testing involves installing the system for experimental purposes and testing it against daily working. In some cases, pilot tests are done primarily in house before deploying it in real
environment for real pilot test. This in house pilot testing is called alpha testing. The pilot testing performed by end user is called beta testing. One other approach in acceptance testing is to deploy the new system in parallel with old system. The advantage of this approach is if new system fails to meet user requirements then user can immediately switch to old system. User’s everyday working will not be affected in case of any system failure. For more information about acceptance testing, please refer to reference [16]

3.3 Testing modes

3.3.1 Black-box Testing

![Figure 7, Black box testing]

As shown in Figure 7, in black box testing, we do not have any knowledge of code structure, we understand the system only by giving inputs and taking outputs. In black box testing, inputs are given to the system under test and then actual outputs are compared with expected outputs. If actual outputs match with expected outputs then we say that system is performing right function and it has passed the test case. But if actual outputs are not same as expected output then we say that system has failed the test case and correction in system should be made. Black box testing is performed on user requirements and system specification. To perform black box testing, tester does not have to have programming knowledge as he does not go into the implementation details. Tester should only know what system under test should do. Tester gives inputs and takes records outputs without know how system under test is generating this output. Black box testing can be performed on unit level or on system level. For more details about black box testing, please refer to reference [20]

3.3.2 Pros and Cons

Black box testing technique can be used at any level. As the testing level increases, the size of the system also increases and it becomes difficult to perform white box testing. So at higher levels, the black box testing is more suitable. When using black box testing, the tester cannot be sure of how much code he has covered. Or whether a particular block of code is tested or not because he has no access to the code, he cannot see the code.[3] But on the other hand, black box testing does not require a tester to be good in programming.
3.3.3 White-box Testing

White box testing has different synonyms. White box testing is also called as structural testing or clear box testing or transparent box testing. All terms have almost same meaning that is the code is visible to the tester. In white box testing, we have access to the internal structure of the system as shown in Figure 8. We can peek into the code and analyze how code is working. We can see implementation details of a system. In order to perform white box testing, a tester needs to have good programming skills in order to successfully design and execute test cases. Tester selects inputs that execute all necessary code. Which input will execute which code? This information is gathered by examining the code structure of the system under test. After giving inputs, the tester examines the inputs and behavior of the system. This strategy helps to improve the quality of the code by exposing loop holes in the code.

White box testing can be performed on unit or system level testing. White box testing is also performed on integration testing. In unit level testing, white box testing is done to see different paths within a unit. In integration level testing, white box testing is performed to examine paths between different units. For more details about white box testing, please refer to reference [22] and reference [10].

White box testing has two main sub types, data flow testing and control flow testing. Data flow testing concentrates more on the points where values are assigned to the variables or where these values are used whereas control flow testing concentrates more on code that cannot be tested using inspections and reviews. In control flow testing, the testing is based on internal paths and structure of the system. To test how much code is tested, a criterion called code coverage is used. Code coverage can be done on different levels. For example, line coverage, decision coverage and condition coverage. In line coverage, the aim is to execute lines of the code, irrespective of the decision or condition. In decision coverage, the aim is to test the decision for true and false whereas in condition coverage, the aim is to test each condition within a decision. But the problem in condition coverage is that each condition is not tested for both true and false. To overcome this limitation, multiple condition coverage is used. In multiple condition coverage, each condition within a decision is tested for both true and false.

3.3.4 Pros and Cons

By using white box testing, code structure can be improved. As in data flow testing, improper use of variable values can be detected and eliminated. Limitation of control flow testing is that tester should have good programming skills to control flow of the code. Because of the fact that tester has to understand the code, control flow testing becomes very time consuming.
3.3.5 Grey box testing
Grey box testing is in between black box testing and white box testing. It is also called translucent testing. In grey box testing, tester only knows such details of the code which enables him to understand that how a particular feature is implemented. It is not necessary for a tester to know all implementation details. While performing grey box testing, the tester prepares the test cases using black box strategy that is preparation of test cases using requirement specification documents and then analyze particular feature of the system using white box testing strategy. For more details about white grey testing, please refer to reference [4].

3.3.6 Pros and Cons
Grey box testing technique has the benefits of both black box testing technique and white box testing technique. But its limitation is that as there is no full code access to the tester so no full code coverage can be assured by the tester.

3.4 Testing Types

3.4.1 Functional testing
Functional testing is considered as a sub-type of black box testing because we concentrate more on what the system is doing rather than how the system is doing. In functional testing, we do not peek into the implementation details of the system under test. We provide inputs to the system and see the behavior of the system and record the outputs. Then we analyze that the system is performing the right intended functionality.

For example, for this thesis work, when functionality testing of AddCtcID interface was performed. CTC ID was entered and next button was pressed. Expected result was that the system under test should fetch data regarding that CTC ID and text areas of next interfaces should be populated with that data. So this expected result was matched with actual result. If on pressing next button system under test fetches the right and data and populates the text areas of next interfaces with this right data then system is performing its intended functionality correctly.

There are different types of functional testing. For example, boundary testing and equivalence class testing. Boundary testing focuses on the input boundaries of a system because mostly the bugs lie on the boundaries. These bugs can be either in requirements of a system or in the code. Most efficient way of finding these bugs is inspection [6]. Boundary value testing is performed by first identifying equivalence Classes then by identifying boundary of each equivalence class. For each boundary value, test cases are created. These test cases are created by selecting one value on boundary, one value just above the boundary and one value just below the boundary. The aim behind equivalence class testing is to reduce the number of test cases to a manageable size with keeping reasonable test coverage. Each equivalence class contains data that results in same output from the program.

One very important feature of white box testing is code coverage. Code coverage means that how much code is executed when test cases are run. By using this information, particular code segments can be tested. High code coverage requires more test cases. Writing test cases require effort and time. There are several tools in market for measuring code coverage. One of the tools is BullseyeCoverage. This tool is used to measure code coverage of C++ programs.

3.4.2 Pros and Cons
The equivalence Class testing is effective where system takes set of data within a range. It is assumed that all data in one equivalence class is treated same by the system. Boundary value testing and equivalence class testing can be performed on unit level, integration level or system level.
3.4.3 Non-functional testing

In the non-functional testing, the system is tested against the non-functional requirements. The non-functional requirements define quality aspects of a system. If a system is fulfilling all functional requirements but the system is unsecure or the system is very slow then the customer will probably not want such kind of a system. So only implementing functional requirements does not make a system complete. To test non-functional attributes, non-functional testing is used. Following are some of the examples of non-functional testing.

3.4.3.1 Compatibility testing

Compatibility testing is an example of non-functional testing. In compatibility testing we test the system under test in different environments. The aim behind running system under test in different environments is to check how it behaves in different environments. For example for this thesis, system was run in Windows environment and in Linux environment. The difference in behavior was recorded and analyzed. As the system was intended to run in mostly Linux environment and sometimes in windows environment, more emphasize was for Linux environment. It was seen that Functionality was same in both environments only some minor graphical changes were recorded. For example, appearance of text fields, text areas, buttons and alert messages etc were different in Windows and Linux.

3.4.3.2 Usability testing

The aim behind usability testing is to analyze how much user friendly the system is or we can say that how easy it is for the users of the system to perform their operations correctly. Usability testing of a system is done by giving it to its end users. End users of a system test it and give their feedback regarding the system. Feedback of the users is the input for improvement of the system. To increase the usability of software, usability testing is performed. In usability testing, users are given an opportunity to use the system. They are given some tasks that they have to complete. While they are performing those tasks, usability experts observe users’ behavior. Users are encouraged to think aloud. Usability experts then ask some questions after the completion of each task. These questions are for example, how easy or difficult you found it to complete the task? What options you want to add into the software, what options you think are confusing. How you want them to be? After taking the feedback, usability experts prepare their analysis and give their recommendations to the developers about design changes. After the development of new prototype, the prototype is again tested by the users and same process is followed. Improvements are measured and the process is repeated until expected usability level is achieved.

For this thesis, after development of a prototype, automation team was asked to use the system. After their use, their feedback was taken and improvement was carried out for the next prototype. For example, there was a Execute Query button on AddCtclID interface as shown in Figure 37. When the prototype was given to automation team for usability testing, one of the automation team members gave feedback that this button should be removed and functionality of this button should be put in Next button. This feedback was discussed with automation team lead and after his approval, this change was implemented in next prototype.

3.4.3.3 Pros and Cons

There is a general perception that usability testing is not necessary, it requires complex and expensive activities. But in reality, usability activities are expensive but they pay off. It is proved in various cases that usability increases the sale, reduces maintenance and redesign costs, it reduces user support costs and improve brand name. [14]
3.4.4 Regression testing
Regression testing is done to uncover new bugs that may have introduced during the implementation of new features. The aim of regression testing is to verify old functionality after a new release or a new prototype. For example in this thesis work, regression testing of all old functionalities was performed in a newly developed prototype. Old functionalities were tested and it was ensured that old functions are working as they are intended to work and new functionality has not affected old functionality.

3.4.5 Progression testing
Progression testing is performed to test new functionality after the system has undergone a new change or an update. In this testing, we do not test old functionalities. We only test new functionalities. Our aim is to uncover defects in newly implemented features. For example for this thesis, when a new interface was added to the tool, in progression testing, all functionality of that newly added interface was tested. Interfaces already present in the tool were not tested while doing progression testing.

3.4.6 Automation testing
The goal behind automation testing is to find bugs effectively, efficiently, quickly and as cheap as possible. Main concept behind automated testing is that there is an application which executes Software Under Test (SUT). That application gives proper inputs and compares each actual output with expected output. After writing a test suite, the test suite can be run without any human intervention. A test suite can be run manually or automatically. After the execution of a test suite, produced results are examined. Results usually provide information about passed and failed test cases.

Code based test inputs can be generated by using code coverage information. If code coverage information shows any unexecuted code segment then such kind of inputs can be given to the program which executes that particular code segment. Second, in automation testing, interface based test inputs can also be generated. For example, If the tester has to find broken hyperlinks in a web page, a test can be made in which each link is clicked and checked whether it is broken or not. A well-known tool Quick Test Professional provides this functionality. Third, test cases can be generated based on specifications. For this purpose, specifications should be in a format that a tool can read. A tool can read specifications and generate boundary values, valid and invalid equivalence classes or expected outcomes.

Test suites are helpful in various aspects. Some of them are as follow:
- Test suite should be run to verify even a minor change.
- No need to manually test every feature of software after each change.
- All test suites are run before a new release.
- If software behavior is different in different environments, test suits should be run for each environment.
- After implementing a new functionality, test suite for that functionality should be written. This provides initial testing of the code.
- Test suite not only executes the software but also it is responsible for setting up the environment for the software and after execution, clearing the environment.

In Ericsson, automation team is currently working on a project containing eighty five test cases. This thesis work is to develop a tool that can produce a java file that will perform automation testing. So we can say that, the output of this tool will be input for the test automation.
3.4.7 Pros and Cons
Automated test cases can perform tedious jobs like clicking on each link on a web page and check whether is it broken or not or any button on a web page is working or not. The tests that are rerun can be automated and rerun without any human intervention. The quality of testing process can be improved by defining a proper standard of the test cases and generate automated test cases using that standard. Human may miss some steps during the testing which compromises the quality of testing but by automating test cases, this danger can be minimized. If a test case is written properly and that test case is completing all required steps then testing quality will not be decreased. That complete test case can be run again and again without any worry of missing any step.
The drawback of automated test cases is that people usually have very high expectations that by using automation testing, many new defects will be uncovered. But in reality, the automated tests can contain deficiencies. There is a possibility that automated tests are badly designed or written. So depending fully on automated tests can lead to undesirable results. One other drawback of the automation testing is maintenance. Maintenance of automated tests is costly.

3.5 Hardware and tools
Figure 9 shows how hardware is booked. Automation team is a sub part of Feature Verification department, when automation team has to use any hardware, they see whether the hardware which they want to use are available or not. They see this in a booking tool used by feature verification department. Anyone from feature verification department who wants to use hardware has to book it in the booking system. Then automation team generates a ticket for configuration of specified hardware. Configuration of hardware is responsibility of ITTE team. They configure hardware. After configuration, Automation team can use the hardware.

![Figure 9. Hardware access process](image)

Test cases are run using one or more than one hardware. When a test case is run, help methods are added to set up hardware or after using reset the hardware. These methods contain code to connect to the specific hardware. Give input data and then takes back the result.
All hardware are used to simulate real network entities. Names of the hardware used by automation team are given below.
1. CCN
2. AeroFlex
3. LTESim
4. PropSim
5. OSS
6. Real UE
7. ENodeB
3.6 **JCAT framework**

JCAT framework is used to test java based applications. JCAT framework consists of several parts and is based on Open Source software.

Following are some advantages of JCAT:
- Java based development environment.
- Test cases executed directly from Eclipse.
- Rapid Test Case development with debug features.
- Reduced execution time.
- Open source framework, community driven maintenance and support.

3.6.1 **JCAT Layers**

As shown in Figure 10, there are six different JCAT layers. Automation team works in Test Case layer. Automatic Test Builder will generate java file that will be run in Test case layer.

![Figure 10, JCAT Layers](image)

In the thesis work, Subversion is used for version control and Maven is used for management of project’s build. More information about Subversion and Maven is given below with other parts of JCAT.

There are two testing frameworks in Ericsson, first one is Generic Test Environment (GTE). The GTE was developed by Ericsson in erlang, a language developed by Ericsson. The GTE is used by Design and Development department for white box testing. The automation team performs black box testing. The hardware management using GTE is extremely difficult, JCAT provides an ease to use hardware. So automation team uses JCAT as testing framework.
3.6.2 **JUnit 3**

“JUnit is a simple framework to write repeatable tests. It is an instance of xUnit architecture for unit testing frameworks.” [19]

3.6.3 **Subversion**

Subversion is a very popular version control system. It is an open source system founded in year 2000 by CollabNet. It was developed as a project of Apache Software Foundation. Figure 11 shows architecture of subversion.[2]

![Subversion Architecture Diagram](image)

**Figure 11, Subversion architecture**

3.6.4 **Eforge**

Eforge is a home for developers within Ericsson who are collaborating on code, who want to work in a new, agile way, or who want to publish reusable software components for other Ericsson engineers across the entire Ericsson Group to use. Eforge provides a set of tools that are well known in the open source community, such as source code revision control, mailing lists, bug tracking, message boards/forums, task management, and total web-based administration for project owners. [5]
3.6.5 Maven

“Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.”[1]
3.6.6 **Hudson**

Hudson is used for Build Management. Hudson has been configured to check for new commits every thirty minutes and run a new build if anything has changed. If the build is successful, any Unit tests found in the project will also be executed.

Hudson is an open source continuous integration (CI) server. A Continuous Integration server can perform following tasks [5]

- Commit source code
- Building the project and then testing the project
- Publishing the results
- Deliver results to specified team persons

Figure 13 shows architectural overview of Hudson.

![Figure 13, Architecture Overview](image-url)
3.6.7 **Sonar**

Sonar is an open source platform to maintain quality of the code. Sonar incorporates seven different aspects of quality of code. [18]

![Diagram of seven aspects of quality of code]

Figure 14, seven aspects of quality of code

After build and tests have been successfully run, Sonar takes over and checks the code for more than six hundred coding rules, unit test code coverage, and all classical metrics related to Lines of Code, Cyclomatic complexity, Duplicated code and Comments.[18]
3.7 **Information model**

Figure 15 shows information model of the entities used in this thesis report. A brief explanation of how these entities communicate with each other and a short description of these entities is given below.

![Diagram showing information model](image)

**Figure 15, Information model**
3.7.1 **FV Legacy team**
Legacy team is responsible for:
- Creating test cases.
- Execution of Legacy CTC:s
- Reviewing code and TC description, handover, mentor roles.
- Automation of CTC backlog.
- Execution of automated test cases developed by automation teams.

The test cases that are run again and again are delivered to automation team by Feature Verification (FV) team for automation. Automation team automates the test cases and legacy team run those automated test cases. As shown in Figure 15, legacy team directly communicates with automation team. Legacy team uses test cases by having access to work packages.

3.7.2 **Automation team**
Automation team automates test cases given by FV team. These test cases are put in work packages. Figure 15 shows communication of automation team with legacy team. Automation team automates test cases and put them in work package.

3.7.3 **Work Package**
Work Package is a collection of different TC suites.

3.7.4 **TC suite**
The whole java file can also be said as TC Suite and the TC suite includes one or many CTCs.

3.7.5 **TC**
TC is abbreviation for Test Case. TC is a collection of different CTCs. A TC belongs to a TC suite. One TC can have one or more than one CTCs. Relationships of TC are shown in Figure 15.

3.7.6 **CTC**
CTC is abbreviation for Configured Test Case. One CTC can belong to only one TC. There can be one or more CTCs belonging to one TC as shown in Figure 15.

3.7.7 **TC Header**
Java file contains a description (TC Header) of the whole TC suite; this description is fetched from database. TC Header contains CTC ID, TC ID, TC heading, TC details, System Function Group, CTC heading, CTC details, quality level and configuration values. All above information is already stored in the database in different columns of different tables. Previously, the tester has to connect to the database, execute a query and fetch these column values from each table. Then he used to copy this information and paste it into the java file under TC header. Now, this job is being done by Automatic Test Builder. For more information about TC header generation, please read section 5.1.3 and section 5.2.

3.7.8 **Preamble**
Preamble part includes all methods required to setup environment for executing test case, which includes following steps:
- All variables are initialized.
- Hardware configurations are done.
- Resources required to execute test case are reserved.
3.7.9 **Post amble**
Post amble part includes all methods required to restore the environment. For example free the memory reserved in preamble phase, restore attributes reserved in preamble phase and restore the environment to the original state.

3.7.10 **Help methods**
Help methods are the methods which are used in test case class. For example methods used to setup the environment and restore the environment.
In automatically generated java file, implementation of these methods is moved to other classes and methods are called by creating an object of that class.

3.7.11 **Main execution TC methods**
In main execution TC, tester writes the code which will be executed in order to run a test case. After running the code for each CTC, a verdict is returned. Verdict tells whether the test is pass or fail. For each CTC there is a separate execution method.
For example of main execution TC methods, please see section 8.3.

3.7.12 **Test methods**
These methods are the implementation of Main execution TC methods. Tester calls test methods in main execution TC method. For example of test method, please see section 8.4.

3.7.13 **Signum**
Signum is an employee ID.
4 Implementation

4.1 Overall structure of the application

Figure 16 shows the overall package structure of the thesis work. All thesis work is under one package named automatic_code_generator_src. automatic_code_generator_src package has five sub packages named as hardware_files, images, SFG_files, user_interface, working_code. Hardware_files package contains all text files. Each text file contains all imports, member variables, preamble part, postamble part related to one hardware. So in total there are seven text files containing all data related to seven different hardware. If automation teams decide to add more hardware, they can simply go in hardware_files package and add more text files containing all necessary data of hardware. This data will be put into the generated java file if specific hardware is selected. Similarly, SFG_files package contains all text files related to each SFG. At this time, there are twenty two different SFG. So there are twenty two different text files. User_interface package contains classes for user interfaces. As there are seven user interfaces. So there are seven different classes. Working_code package contains two classes. CodeGenerator class contains different methods that are used in interface classes and PanelHistory class is used by MainPage class.

![Figure 16, Package structure](image-url)
Figure 17 shows class diagram of all classes in the thesis work project.
4.2 Screen shots of application

Above is the first screen that comes when a user runs Automatic Test Builder. For the thesis work, there is only one option to search by CTC ID, but more search criteria can also be added. E.g. Search by TC ID, Search by Work Package. For this reason, size of this interface is bigger to accommodate future additions. For more information about the future improvements, please see section 6.5.
In this interface, user can add or remove CTC IDs. When user is finished with entering CTC IDs, he will press next button. On pressing next button, data regarding each CTC ID will be fetched from database and text fields on next interfaces will be populated. When user enters a CTC ID, he can check automated and regression statuses of CTC ID by pressing enter button while keeping the focus in the text field. These statuses help user to decide whether to automate entered CTC or not. If CTC ID is not valid, on pressing enter button, alert message will come showing that CTC ID is not valid, please enter valid CTC ID.

There are two checks implemented on this interface.

1. User cannot delete first row.
2. User cannot move to next or last interface without entering at least one CTC ID.
3. If user enters same CTC ID more than once, alert message appears specifying duplicate CTC IDs and asking to remove duplicate occurrences. Without removing duplicate occurrences, data is not fetched from database.
Figure 20, with multiple CTCs

Screen shot shows multiple CTC IDs.
Screen shot shows information fetched from database regarding TC ID. User can edit this information.
Screen shot shows information fetched from database regarding CTC ID. User can edit this information.
In this interface, user has to enter TC name, the name of java file which will be generated will contain TC name and _TestCase appended at the end. User can also change file name in next interface. After entering TC name, user can select hardware which he wants to use for the test case.

There are two checks implemented on this interface.

1. User must enter TC name.
2. TC name must contain Work Package number at the end of TC name.
Screen shot shows volition of second check.
In this interface, Signum of user will be displayed in Signum field. User can also change it. User will select the location of java file. After selecting location, user can press on Generate java File button and java file will be created on specified location. There are two checks implemented on this interface.

1. Signum field cannot remain empty.
2. User has to select file location. Generate java File button will remain disabled until user selects file location.
Figure 26, GenerateFile

Screen shot shows Browse window after user has clicked on Browse button.
Figure 27, GenerateFile
Screen shot shows confirmation message after Generate java File button is clicked.
4.3 **Structure of java file**

![Diagram of java file structure]

Initial structure of the java file which is generated by Automatic Test Builder is shown in Figure 28.

Initially it was decided that implementation of Help methods will be in java file generated by Automatic Test Builder but this decision was changed later and new requirement was that a new class hierarchy should be created and help methods implementation should be moved to those classes. If any help method needs to be added then that method should be called by creating an object of class containing specified help method. So this change was implemented and structure of java file was changed. Figure 29 shows final structure of the java file.
Figure 29, Final java file structure
4.4 **Class diagram of java file**

Figure 30 shows class diagram of java file created by Automatic Test Builder. In Figure 30 TestCase class is the java file created by Automatic Test Builder, it is inherited from LteBaseTestCase class. Implementation of all help methods are in HardwareMethods class, SFGMethods class and their child classes. Any help method that a tester wants to use in the test case, he can call it from any of these classes.

![Class diagram of java file](image)

*Figure 30, Class diagram of java file*
4.5 Inputs and output of the application

Figure 31 shows all inputs to the Application. Tester enters CTC ID then after the data is fetched from database and displayed on different interfaces, he can change any information. Then on Hardware interface, he has to enter test case name and select hardware. He also has to enter signum in generate file interface and specify the java file path.

1. CTC ID.
2. Any change in TC or CTC information.
3. Test case name.
4. Hardware.
5. Signum.
6. Path where java file will be saved.

4.6 Manually created java file VS Java file created by Automatic Test Builder

In manually created java file, tester has to create a class with the name of test case, Write header TC (description of test case class) then we has to add all imports, member variables, call all help methods used in preamble part, all help methods used in post amble part. Then he has to write main execution TC methods. He also has to implement test methods and help methods. Where as in the java file created by Automatic Test Builder, All above mentioned parts are add automatically. Tester just has to validate these parts. If he thinks that any code is missing, he can add that code or if he thinks any code is extra, he can remove that code. But java file provides a complete structure of whole test case. One main difference in manually created java file and java file created by Automatic Test Builder is that in manually created java file, implementation of help methods is included where as in java file created by Automatic Test Builder, implementation is moved to other classes. Methods are called from those classes. For more information regarding structure of java file, please see section 4.3.
5 Development process
The task was assigned to me of developing such a tool that can collect different parts of a test case from different sources and based on tester’s selection, assemble all parts in the right order and create a java file.
Thesis work was started on February 13th, 2012. The first task was to develop organizational understanding by reading all necessary organizational literature such as “JCAT_RAN_getting_started” and “Test_Design_User_Guide”. Then a presentation was delivered by the automation team lead to give a clearer idea of what they are expecting from this thesis work. Then the task was to develop a time plan for the whole thesis work. This time plan was not final but it was just to create an abstract view that will help to organize the development activities. This time plan was developed keeping in mind current requirements. Time plan was sent to the Line Manager. The Time plan is included in section 8.7.
Then there were discussions with team lead to elaborate the requirements.
After these discussions, Development was started. The development process was divided into following stages.

5.1 Semi-automated process

5.1.1 Generation of text file
First task was to read data from database and store it locally so that the data can be used for further processing. It was decided by the automation team lead that they require data of nine columns from database. This data will be added into the TC header.
As in section 6.1, it is explained that automation team does not have direct access to the database. So they can only fetch information using a tool ClearQuest. So, how to get required data from database using eclipse?
This was a big question mark which was to be answered. Several meetings were arranged with the team lead automation to discussed possible solution of connecting to database using eclipse.
In meetings, team lead automation suggested that as there is a functionality of ClearQuest that after executing a query, a text file can be generated containing query results. This text file can be used for the input. It was decided that this option will be used and in parallel, meetings will be arranged with the development team and efforts will be made to get direct access to the database.

So, for the first step, the tester will generate a text file manually and save it on a location and Automatic Test Builder will read that text file and process the data in it. Finally, after all processing, a java file will be created. Figure 32 describes the structure of fetching data from database through ClearQuest, generating a text file and then giving text file as input to the tool. After tester edits the information, java file will be created.
5.1.2 Reading text file and saving data in local database

Now the second step was to read text file and then save data in a local database. The text file contained all nine column values regarding each CTC ID separated by a delimiter “;”. Each column value was separated and saved in a local variable. For the code review, please see section 8.2.1.

After reading data from the text file, next task was to save information into the local database. So that information can be used for further processing. For that purpose, design of the local database was created and discussed with the Team Lead of automation team. After agreement, the database was created. Mysql was used as database management system.

After the successfully creation of database, code was written to put all information into local database. For the code review, please see section 8.2.2.

5.1.3 TC header generation

TC database results were mainly used in TC header creation. Series of meetings with automation team were conducted to finalize which column value should be put into which TC header part. After agreement, a complete code was written to get information from local database and then put each column value into the specified TC header part.

After completing the semi-automatic process, first prototype was presented to the automation team. Figure 2 shows first prototype.

5.2 Fully-automated process

![Figure 33, Fully-automated structure](image)

Figure 33 exhibits the fully automated process. Information regarding each CTC ID is fetched from the database using a SOAP service by the tool. After tester edits the information, a java file is generated. Fully automated process was implemented in Prototype 2, it was the replacement of reading data from the text file.

5.2.1 Connection to database using SOAP service

As discussed in section 5.1.1, it was decided to have continuous contact with development team so that any solution to establish a connection between eclipse and the database can be attained. After posting question on Ericsson’s internal forum, contacting different Ericsson’s department members, arranging meetings with development team members. Finally, two solutions were proposed.

1. xml dump by sending a url request. CTC ID, signum and password will be embedded within the url.
2. String of column values by using SOAP service developed by Ericsson. CTC ID and all column names will be provided as input arguments.
The first option was not an efficient solution because xml dump contains all column values regarding each CTC ID but we only needed nine column values. So, fetching all column values from the database will increase load on the database server. So this solution was not considered. Second solution was efficient and viable because only required data was being fetched from the database minimizing load on the database server. Second advantage was that the resulting data can be directly stored in a string variable. So no parsing of the data was needed, which increased the efficiency of the tool.

Now next step was to study the SOAP service so that by using it, data can be fetched from the database. After studying the SOAP service, it was concluded that there are some methods provided by the SOAP service that can be used to establish connection to the database and then on giving column names and CTC ID, column values can be fetched from the database. So after studying the SOAP, algorithm was designed to fetch data and then save it and then use it in further processes.

5.2.2 Java file generation
Finally, text file generation, text file parsing, saving information in the local database, fetching information from the local database was replaced by new module.

5.3 GUI Design
After completion of the main back-end development, GUI development was next stage. In this stage, the first step was to have discussions with automation team, understand their needs and then prepare the mockups. After discussions, the mockups were prepared. After approval from the Team Lead automation, GUI design was started. Design was based on the mockups. Snapshots of the mockups are shown in section 8.1. Each mockup represents an interface of Automatic Test Builder.

5.4 Working GUI
After designing the GUI, next phase was to embed code into the GUI so that it performs desired functionality. Some code was already written, for example fetching data from the database and putting information into the TC header. Other code was written in this phase of making working GUI. Working GUI was implemented in prototype 3. GUI was continuously improved for all subsequent prototypes. Code structure of the java file was firstly implemented in prototype 4 and was continuously improved in all subsequent prototypes.

5.5 Adding main execution TC methods and help methods
Adding the help methods was comparatively difficult than adding main methods. This section discusses more about adding the help methods. Regarding information about the main methods, please refer to section 3.7.11.

Concept of adding the help methods is that any method that will be used in setting up the environment or clearing the environment or any method used in the main execution TC part will be added here.

Before the development of this tool, each test case had complete implementation of all help methods within a java file. If any change occurs in the implementation of help method, all test cases containing that help method will have to be changed manually. This creates a huge workload. This problem was resolved by developing a new class for the help methods. All help methods were implemented in that class and were called in the java file. This eliminated the problem of having different implementations of single help method. Main execution TC methods and help methods were firstly implemented in prototype 5.
5.6 **Creation of help methods class hierarchy**

After the creation of new class for help methods, requirement was once again changed that with the passage of time, number of help methods will increase. So it will become extremely difficult to search any particular help method or change it if they are in a single class. This will create huge difficulty to maintain this one class. So help methods should be divided in such a way that maintenance of help methods becomes easier. So it was then decided by the automation team to have a completely new class hierarchy of help methods.

5.6.1 **Creation of Hardware methods class hierarchy**

So after discussions, it was decided that separate classes will be created for each hardware and all hardware classes will be inherited from a parent class containing common methods used by all hardware classes. Figure 34 shows class hierarchy of hardware methods. Hardware methods class hierarchy was implemented in prototype 6.

![Hardware Methods Class Hierarchy](image)
5.6.2 Creation of System Function Group methods class hierarchy

After agreeing upon creating hardware help methods class hierarchy, creation of SFG (System Function Group) class hierarchy came under discussion. There are twenty-two different SFG, so following same approach as hardware methods class hierarchy was not a good idea. Creation and maintenance of seven classes is much easier that creation and maintenance of twenty-two different SFG classes. So after several discussions, it was decided that division of SFG classes will be made on other criteria for example, all methods relating to throughput will be put in one class. Then all these classes will be inherited from one parent SFG methods class. For this thesis work, only one class named “Start and Restart RBS” was created just to give guidance to the testers so that they can add new classes in the future by themselves. There was a possibility that the SFG parent class can contain some hardware methods that are in hardware parent class. So it was decided that SFG parent class will inherit those common methods from hardware parent class. Figure 35 shows class hierarchy of SFG methods. In Figure 35, child classes of SFGMethods class are just to illustrate future creation of classes. SFG methods class hierarchy was implemented in prototype 7.

![Figure 35, SFG help methods class hierarchy](image-url)
5.7 **Documentation development**

One of the requirements was development of documentation about Automatic Test Builder. This all documentation should be available on Ericsson’s internal wiki pages. List of documents that were prepared is:

- Manual about Automatic Test Builder
- Tutorial on how to start Automatic Test Builder
- Tutorial on how to run Automatic Test Builder and generator java file.

5.8 **Training of testers**

Automatic Test Builder was on SVN so when development was complete, it was available to automation team to test and use it. Several demonstration sessions were arranged to show the working of Automatic Test Builder. Testers were encouraged to give their feedback so that the quality of Automatic Test Builder can be improved. Tutorials were developed to help the testers in running Automatic Test Builder without any problem.
6 Discussion

6.1 What they did before

Before the development of this tool, automation team sued to automate test cases manually in JCAT environment. They have to write Header information of each CTC so that tester can know about the test case by reading header information. They also have to create a test class, write preamble, post amble, help and main test methods for each test case they automate. Writing this all code was a very hectic and timing consuming. There was no defined standard for writing a test case which incorporated inconsistencies.

If any information is needed from database, a tool named ClearQuest is used to fetch information from the database. Information is fetched by executing queries. By using this tool, a tester can add, delete and update any information of Work package, Test Case, Configured Test Case. Tester does not have direct access to database. He can only access database using ClearQuest. ClearQuest provides a functionality to create a text file of the results fetched from database.

6.2 What they can do now

The FV legacy team tests each hardware by executing test cases generated by Automatic Test Builder. So, the level of testing performed by the FV legacy team using Automatic Test Builder is unit testing. The testing mode for which the test cases generated by Automatic Test Builder are used is Black box testing as generated test cases only tests the functionality of each hardware by giving inputs and comparing expected outputs to the actual outputs. The test cases generated by Automatic Test Builder are not aimed to test how a particular hardware is performing the required functionality. FV legacy team is only concerned about what a particular hardware is doing. As discussed earlier that the test cases generated by Automatic Test Builder only tests the functionality, so the Testing type is functional testing.

Using this tool, a tester just enters CTC ID and all CTC details are fetched from the database. Then he can edit the information if he wants to and finally a java file is automatically generated. A complete structure of test case class with all preamble, post amble, help methods and main test methods are generated. He then adds or removes some code and run test case class in JCAT environment. Effort and time is saved.

6.3 Testing of the system

Testing was performed by me throughout the development process. Before the completion of a prototype, complete testing of all implemented features was carried out by me. After showing the prototype to the automation team, usability testing of the prototype was carried out. Changes suggested after usability testing and new features were implemented and then regression and progression testing were performed. For more information about testing techniques used in this thesis work, please see section 3.1. After testing, new prototype was shown to automation team. So testing was done for all prototypes throughout the development process. This continuous testing strategy resulted in a high product quality.

6.4 Estimations of experts about the application

According to Team Lead Jonas Widén “It takes about two weeks to write automated test case”. When it was asked that how much time it will take after generating java file from Automatic Test Builder, he said “It will take one week”. So considering Jonas’s opinion, Automatic Test Builder will save 50 percent of the total time spent in writing automated test case.
6.5 Future Improvements

This tool is the first internally developed tool that automation team will use to speed up their automation process. Automation team can use this tool as a base and add more functionality in this tool to make it more sophisticated and intelligent.

Some future improvements can be for example, in the very first interface which provides the option to search by CTC ID. More search options can be provided. Like, Search by TC ID, Search by Work package. For adding each search criteria, all interfaces related to this search criteria, will be developed.

Other improvement can be to add functionality in Hardware interface to add more than one instance of each hardware because for this thesis work, tester can add only one instance of hardware.

6.6 Reflection on development method

It is a general perception that in the beginning of the software development process, client does not know what he really wants. With the passage of time, he changes his requirements. Keeping in mind this phenomenon, there should be a development method that can deal with requirement changes efficiently. In the beginning of this thesis work, the client and end user that is automation team was not hundred percent clear about what they finally want. I wanted that I should communicate with automation team as much as possible so that I can understand their requirements better. I also wanted that I involve automation team by getting their feedback throughout the development process so that I could develop the right system for them. I searched for a development method that incorporates all of my requirements. The prototyping method was the best option for me. I started working using prototyping as the development method. I used to show automation team every prototype that I developed, I used to discuss each functionality before implementation. I used to take their requirement and then suggest my solution and give them a freedom to decide what they want. After their decision, I used to implement that particular functionality. Soon after the implementation of that functionality, I used to show them the implemented requirement so that if they want, they can even more improve that functionality. Sometimes I had to do extra work because of change in requirement but by always welcoming their requirements, I noticed an increase in satisfaction level of automation team.

One more important factor to successfully incorporate the changes was that I tried to reduce the coupling (inter dependency of different modules) so that in future if I had to change (add or remove) some functionality, other parts of the system would not be affected. For decoupling, I made a separate class named CodeGenerator that contained all methods that were used in the system to perform different operations. If I had to add any new functionality, I added one more method to fulfill that functionality. I also reused already written methods to implement new functionality. This allowed me to save my efforts from writing the code from scratch to implement new functionality. I would also like to mention that I got a great support from internet in finding different solutions. I did not prefer to reinvent the wheel.

In the end, I would like to say that if you work continuously and have faith in Almighty God, You will be successful.
7 References


8 Appendix

8.1 Mockups

Figure 36, Interface for Search by CTC ID
Later on it was decided that Execute Query button should be removed and functionality of Execute Query button should be put in Next button.
Figure 38, Interface for displaying TC information
Figure 39, Interface for displaying CTC information

Later on it was decided that CTC Verdict should be removed because CTC Verdict is not necessary information for TC header.
In the beginning, it was decided that there should be an option to import a text file containing names of hardware. So Load names button was added in above interface but later on text file containing hardware names was moved to the package on Automatic Test Builder so now there was no need to externally select a text file. So Load names button was removed.

Later on it was also decided that functionality of Ok button should be moved in next button. So Ok button was removed and functionality was moved in Next Button.
Figure 41, Interface for java file generation
8.2 Code Snippets

8.2.1 Reading text file

```java
public void readFromFile(){
    try{
        String[] colValues = new String[totalNumberOfCol]; // Used to store the values of string file
        // totalNumberOfCol is 9.
        int colNum=0; // Used to track which column is so that we can paste right value in right column in Database
        FileInputStream fstream = new FileInputStream("c:\Users\eabbsaa\Documents\autoFile.txt");
        DataInputStream in = new DataInputStream(fstream); // Get the object of DataInputStream
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        String strLine,newStr="";
        int indexOfColon=0,counter=0,preIndex=0;
        while (strLine != null) { // Read File Line By Line
            counter=0; // Counts how many times ';' occurred in a line(StrLine)
            // Boolean lastDataOnLine=false;
            System.out.println (strLine);
            for(int i=0;i<strLine.length();i++) // Count how many times ; has occurred in the string strLine
            { // If there will be data then newStr.length() !=0 and if there will not be a colon then we will not do colNum++
                if(strLine.charAt(i)==';')
                    counter++;
            }
            System.out.println(counter);
            /** * If there will be data then newStr.length() !=0 and if there will not be a colon then we will not do colNum++ */
            for(int i=0;i<counter;i++) // If there is data and also there are colons and there is no data but there are colons (Columns with no value)
            {
                indexOfColon=strLine.indexOf(";", indexOfColon)+1;
                System.out.println(indexOfColon);
                if(i==0) // to handle new line, previous line data will be appended to new line data
                    newStr=newStr+"\n"+strLine.substring(0,indexOfColon-1);
                else
                    newStr=strLine.substring(preIndex,indexOfColon-1);
                preIndex=indexOfColon;
                System.out.println(newStr);
            }
        }
    }
```
colValues[colNum]=newStr;
    colNum++;

    newStr=strLine.substring(preIndex,
    strLine.length());
    if(newStr.length()!=0 &
    counter!=0){ //There is data
    //if(lastDataOnLine==true)
    colValues[colNum]=newStr;
    }
    else if(newStr.length()!=0 &
    counter==0){ //Data but
    no colon, to handle data like ===== or ------
    colValues[colNum]=colValues[colNum]+
    \t\n+newStr;
    newStr=colValues[colNum];
    }
    if(colNum==totalNumberOfCol-1)
    {
    saveToDB(colValues);
    colNum=0; //when we reach last column, reset the
    column index to 0
    }
    //System.out.println(newStr);
    }
    in.close(); //Close the input stream
} catch (Exception e){ //Catch exception if any
    System.err.println("Error: " + e.getMessage());
    }
### 8.2.2 Saving data into local database

```java
public void saveToDB(String[] colValues) {
    String dbUrl = "jdbc:mysql://localhost/test_db";
    //String dbClass = "com.mysql.jdbc.Driver";
    String query = "insert into headerinfo"
    + " Values(""'
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection(dbUrl);
        Statement stmtStatement = con.createStatement();
        int val = stmtStatement.executeUpdate(query);
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
    catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

### 8.3 Main execution TC methods example

```java
@Override
protected Verdict execution() throws Throwable {
    // This line is to enter comments in log.
    setTestStep("execution start");
    if(isTmRunning("LTE00000001")
        executeLTE00000001();
    // This is Main execution TC method for CTC ID LTE00000001. All help methods and other
code are written in its implementation i.e. Test Methods.
    if(isTmRunning("LTE00000002")
        executeLTE00000002();
    // This is Main execution TC method for CTC ID LTE00000002. All help methods and other
code are written in its implementation i.e. Test Methods.
    if(isTmRunning("LTE00000003")
        executeLTE00000003();
    // This is Main execution TC method for CTC ID LTE00000003. All help methods and other
code are written in its implementation i.e. Test Methods.
    executeLTE00000003();
```
if(isTmRunning("LTE00000004")
// This is Main execution TC method for CTC ID LTE00000004. All help methods and other
code are written in its implementation i.e. Test Methods.
executeLTE00000004();
//This will return verdict whether this test is passed or fail.
return getThcVerdict();
}

8.4  **Test method example**

private void executeLTE00000550() throws Throwable {
// This line is to enter comments in log.
setTestStep("executeLTE00000550");
//Here tester adds all code which he wants to execute to test CTC ID LTE00000550.
    // TODO
}

8.5  **Manually created java file**

TC Suite

package com.ericsson.ate.lte_ran_iov.feature_verification.automation_12B.testcase;

import java.util.ArrayList;

TC Header
/**
 * CTCs covered by this test case
 * LTE00000001 - ANR GSM DRX sleep period assignment - STP4
 * *
 * --- IMPLEMENTATION ---
 * *
 * LTE00000001
 * Short description about will be tested in LTE00000001.
 * *
 * --- VERDICT ---
 * *
 * LTE00000001
 * PASS is set if the number of successful CGI measurements exceeds 90% for any of
 * the following DRX periods 256ms, 320ms, 512ms, 640ms, 1024ms, 1280ms, 2480ms,
 * 2560ms.
 * *
 * */
--- RESOURCE PROPERTIES IN TM4LTE---
* 
* configuration I&V_Full
* ue1 ue li liaeroflex013
* enb1 enb li lienb0510
* enb2
* pgw cpg ln cpg7
* isp1 isp ln lniperf001
* ltelogtool1
* mme1
* enb3
* oss1
* stp
* portbase 0
* eclipse.workspace.path default
* enb3communicator Amos
* jcat.loglevel info
* enb1communicator Amos
* enb2communicator Amos
* view ate_default_atelte
* runtype DEV
* verdictreporting All CTCs
* 
* --- CTC PROPERTIES IN TM4LTE---
* 
* ANRGSM_DRX_SleepPeriodAssignment_WP1898__targetgerancellid 100
* 
* 
* @author emicile
*/

public class ANRGSM_DRX_SleepPeriodAssignment_WP1898_TestCase extends LteBaseTestCase
{
  /**
   * CTC ID for WP1***
   */
  public static enum CtcId
  {
    /**
     * ANRGSM_DRX sleep period assignment.
     */
    LTE00000001
  }
  private Moshell moshellSourceEnb;
  private HandlerManagedObject handlerManagedObject;
private String targetGeranCellId;
private UeNasMode ueNasMode;

Preamble

@Override
protected Verdict preamble() throws Exception {
    setTestStep("preamble");

    targetGeranCellId = getTestCaseProperty("TargetGeranCellId");

    // Moshell setup
    CommunicatorOptions communicatorOptions = new CommunicatorOptions();
    communicatorOptions.configureCompleteMom("1"); // Enable complete mom
    communicatorOptions.useDefaultSetup();
    communicatorOptions.useConnectWithDetect();
    moshellSourceEnb = LteRm.enb1.generateConfiguredMoshellSession(communicatorOptions);
    handlerManagedObject = new HandlerManagedObject(moshellSourceEnb);

    // Start undo mode (for cr/del/rdel/set/bl/deb/acc commands only)

    // Create the needed parent MOs for ExternalGeranCell and GeranCellRelation and
    // make sure that GeranFrequencyGroupRelation exists (precondition).
    createNeededMOs();

    // ueNasMode setup
    ManagedObjectService moService = LteRm.enb1.provideManagedObjectService(LteRm.enb1.getDefaultMoshellSession());
    EUtranCellFDDMO cellMO = moService.getEUtranCellFDDMO("ENodeBFunction=1,EUtranCellFDD=1");
    UeNasModeBuilder nasModeBuilder = LteRm.ue1.getNasModeBuilder();
    int cellId = EnbUtil.calculatePhysicalCellId(cellMO.getPhysicalLayerCellIdGroup(),
        cellMO.getPhysicalLayerSubCellId());
    int earfcndl = cellMO.getEarfcndl();
    int frequencyBand = FrequencyBandConverter.getFrequencyBand(earfcndl);
    int dlFrequency = (int) EnbUtil.calculateDlFrequency(earfcndl);
    nasModeBuilder.setNumOfUe(1).setFrequencyBand(frequencyBand).setCellIdAll(cellId).setDlFrequencyAll(dlFrequency);
    ueNasMode = nasModeBuilder.build();

    // Set serviceStateAnr to operable
    setServiceStateAnrToOperable();

    // Set anrStateGSM to active
    setAnrStateGSMIsActive();
// Remove any ExternalGeranCell pointing at target cell
// Also removes MOs reserving the ExternalgeranCell, i.e. GeranCellRelations
removeAnyExternalGeranCellPointingToTargetNode();

// Verify that the UE source cell is an eUtranCellFdd.
verifySourceCellIsEutranCellFDD();

// Verify UE is inactive in RAC --> Tester says wait for 2 seconds and the UE is inactive
try {
    sleep(2);
} catch (InterruptedException e) {
    setThcLteVerdictAndPrintToLog(new LteVerdict(INCONC,
        "Thread sleep interrupted. This might cause and/or be caused by an
        unreliable environment." +
        " UE might not have had enough time ");
}
return getThcVerdict();

Main Execution TC
@Override
protected Verdict execution() throws Throwable {
    setTestStep("execution");

    // Execute test of CTC LTE00000001
    executeLTE00000001();

    return getThcVerdict();
}
Postamble

```java
@Override
protected Verdict postamble()
{
    setTestStep("postamble");
    // Stop undo mode(u-) and restore environment
    stopUndoAndRestoreEnvironment(moshellSourceEnb);
    // Disconnect Moshell
    moshellSourceEnb.disconnect();
    setThcLteVerdict(moshellSourceEnb.verifyNoErrors());
    return getThcVerdict();
}
```

Help and Test Methods

```java
// ###################################################################
// ##              TEST METHODS              ##
// ###################################################################

private void executeLTE00000001()
{
    setTestStep("executeLTE00000001");
    // A1 Connect to the RBS using moshell.
    // Moshell should already have been connected in preamble(), just verify
    boolean connected = moshellSourceEnb.isConnected();
    if (!connected)
    {
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001,
            new LteVerdict(ERROR, "Moshell was unexpectedly not connected to the RBS.");
    }
    // A2 Change DRX period. Use the following values, one in each iteration [256, 320, 512,
    // 640, 1024, 1280, 2480, 2560].
    String systemConstantsLine = getLineContainingString(answergetSystemC,
        "SystemConstants");
    String[] split = systemConstantsLine.trim().split(" ");
    String proxyNumber = split[0];
    String ldnSystemConstants = split[split.length -1];
    if (!answerAcl.contains("writeSystConst")
    {
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001,
```

```java
```
new LteVerdict(ERROR, "Could not list command WriteSystConst for MO SystemConstants.");

String[] drxPeriodList = {"256", "320", "512", "640", "1024", "1280", "2480", "2560"};
for (String drxPeriod drxPeriodList) {
    // Set DRX period
    handlerManagedObject.doManagedObjectAction(ldnSystemConstants, "writeSystConst",
    new String[] {"283", drxPeriod});  // TODO Check why undo doesn't work when HW is available
    if (!answergetSystemC.contains("283" + drxPeriod)) {
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001,
            new LteVerdict(ERROR, "DRX period was not correctly set."));
    }
}

// Repeat A3, A4, R6, A7, A8 at least 10 times
for (int i = 0; i < 10 ; i++) {
    // A3 Detach and attach UE to source eNodeB
    int detachAllJobID = ueNasMode.detachAll();
    ueNasMode.waitUntilJobIsDone(detachAllJobID, 120);
    int attachAllJobID = ueNasMode.attachAll();
    ueNasMode.waitUntilJobIsDone(attachAllJobID, 120);

    // A4 Move UE until measurement report mention target cell

    // R6 The S-eNodeB has created the MOs for Geran external cell and Geran cell relation

    // A7 & A8 IF R6 MOs exist Increase number of successful CGI measurements
    // and delete MOs. Go back to A4
    // ELSE Increase number of unsuccessful CGI Measurements and go back to A4

} // A9 IF at least 90% of CGI Measurments are successful --> Test passed
// ELSE Go to A2

setTmLteVerdict(CtcId.LTE00000001, new LteVerdict(PASS));
/**
 * Sets serviceStateAnr to operable
 */
private void setServiceStateAnrToOperable() throws Exception {
    // Try to set serviceStateAnr to operable
    handlerManagedObject.setManagedObjectAttribute("Licensing=1,OptionalFeatures=1,Anr=1",
            "serviceStateAnr", "1");
    LteVerdict verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();

    // If we failed to set serviceStateAnr, throw exception.
    if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS)) {
        String errorMessage = "Could not set serviceStateAnr to operable.";
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
        throw new Exception(errorMessage);
    }
}

/**
 * Sets anrStateGSM to active
 */
private void setAnrStateGSMIsActive() throws Exception {
    // Try to set anrStateGsm to active
    handlerManagedObject.setManagedObjectAttribute("AnrFunction=1,AnrFunctionGeran=1",
            "anrStateGsm", "1");
    LteVerdict verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();

    // If we failed to set anrStateGsm, throw exception.
    if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS)) {
        String errorMessage = "Could not set anrStateGsm to active.";
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
        throw new Exception(errorMessage);
    }
}
* Removes any ExternalGeranCell pointing to target cell. */
private void removeAnyExternalGeranCellPointingToTargetNode()
{
  ExternalGeranCellId"
);

  //Check that at least one ExternalGeranCell exists
  if (resultExternalEUtranCellFDD.contains("ExternalGeranCell"))
  {
    Collection<String> linesContainingExternalgeranCells =
        getAllLinesContainingString(resultExternalEUtranCellFDD, "ExternalGeranCellId");

    // Loop over found ExternalGeranCells
    for (String lineContainingExternalgeranCell linesContainingExternalgeranCells)
    {
      String[] split = lineContainingExternalgeranCell.trim().split(" ");
      String ldnExternalGeranCell = split[0];

      String lineContainingCellIdentity = getLineContainingString(answerGetCellIdentity,
          "cellIdentity");
      String[] split2 = lineContainingCellIdentity.trim().split(" ");
      String cellIdentityValue = split2[split2.length - 1];

      // Check if the cell identity of the target Geran cell matches the
      // cell identity of the ExternalGeranCell.
      if (targetGeranCellId.equals(cellIdentityValue))
      {
        Collection<String> linesContainingReservedBy =
            getAllLinesContainingString(answerGetReservedBy, ">>> reservedBy");
        for (String lineReservedBy linesContainingReservedBy)
        {
          String[] split3 = lineReservedBy.trim().split(" ");
          String ldnToreservingObject = split3[split3.length -1];
          handlerManagedObject.deleteManagedObject(ldnToreservingObject);
        }

        // Remove the ExternalGeranCell
        handlerManagedObject.deleteManagedObject(ldnExternalGeranCell);
        // TODO
      }
    }
  }
}

/**
* Creates the needed parent MOs for ExternalGeranCell and GeranCellRelation. In the
process it
also makes sure that GeranFrequencyRelation exists which the test instruction says must be verified.

```
private void createNeededMOs() throws Exception {
    // Parent MOs needed for ExternalGeranCell
    // +-ENodeBFunction
    // +-GeraNetwork
    // +-GeranFreqGroup
    // +-GeranFrequency

    // GeraNetwork
    handlerManagedObject.createManagedObject("ENodeBFunction=1,GeraNetwork=1");
    LteVerdict verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();
    if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS)) {
        String errorMessage = "Could not create MO ENodeBFunction=1,GeraNetwork=1.";
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
        throw new Exception(errorMessage);
    }

    // GeranFreqGroup
    handlerManagedObject.createManagedObject("ENodeBFunction=1,GeraNetwork=1,GeranFreqGroup=1",
        new String[] {"1"}); //frequencyGroupId
    verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();
    if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS)) {
        String errorMessage = "Could not create MO ENodeBFunction=1,GeraNetwork=1,GeranFreqGroup=1.";
        setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
        throw new Exception(errorMessage);
    }

    // GeranFrequency
    handlerManagedObject.createManagedObject("ENodeBFunction=1,GeraNetwork=1,GeranFreqGroup=1,GeranFrequency=1",
        new String[] {"1","d"}); // arfcnValueGeranDl & bandIndicator
    verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();
    if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS)) {
    }
}
```
String errorMessage = "Could not create MO ENodeBFunction=1,GeraNetwork=1,GeranFreqGroup=1,GeranFrequency=1.";
setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
    throw new Exception(errorMessage);
}

// Parent MOs needed for GeranCellRelation
// +-ENodeBFunction
// +-EUtranCellFDD
// +-GeranFreqGroupRelation

// EUtranCellFDD
handlerManagedObject.createManagedObject("ENodeBFunction=1,EUtranCellFDD=1");
verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();
if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS))
{
    String errorMessage = "Could not create MO ENodeBFunction=1,EUtranCellFDD=1.";
    setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
    throw new Exception(errorMessage);
}

// GeranFreqGroupRelation
handlerManagedObject.createManagedObject("ENodeBFunction=1,EUtranCellFDD=1,GeranFreqGroupRelation=1",
    new String[] {"ENodeBFunction=1,GeraNetwork=1,GeranFreqGroup=1"});
verifyNoErrorsVerdict = moshellSourceEnb.verifyNoErrors();
if (!verifyNoErrorsVerdict.getVerdict().equals(Verdict.PASS))
{
    String errorMessage = "Could not create MO ENodeBFunction=1,EUtranCellFDD=1,GeranFreqGroupRelation=1.";
    setTmLteVerdictAndPrintToLog(CtcId.LTE00000001, new LteVerdict(ERROR, errorMessage));
    throw new Exception(errorMessage);
}

/**
 * Verifies that the UE source cell is an eUtranCellFdd.
 */
private void verifySourceCellIsEutranCellFDD() {
    // MO EUTRANCELLFDD cannot be created on a TDD node. So if one exists it is
    // verified that this is an eUTRANCellFDD cell. An EUTRANCELLFDD MO should
    // have been created in createNeededMOS().
    if (!answerEUTRANCellFDDId.contains("EUTRANCellFDDId")) {
        String errorMessage = "Test case setup phase went wrong. Source cell is not of type
        EUTRANCellFDD";
        setThcLteVerdictAndPrintToLog(new LteVerdict(ERROR, errorMessage));
    }
}

/**
 * Stops Moshells undo mode and runs the restore file generated by the undo mode.
 *
 * @param moshellSourceEnb An Moshell session connected to the eNodeB to be restored.
 */
private void stopUndoAndRestoreEnvironment(Moshell moshellSourceEnb) {
    // Fetch the line containing the undo file command. The line looks something like this
    // "To undo, execute command run
    // /home/emicile/moshell_logfiles/logs_moshell/undo/undo_LIENB0506_120124-101402.mos
    // ($undoCommandfile)"
    String undoFileLine = getLineContainingString(answerStopUndo, ".mos");

    // Look for the undo file command in line
    Pattern pattern = Pattern.compile(".*(run.*.mos).*");
    Matcher matcher = pattern.matcher(undoFileLine);

    // If undo file command found
    if (matcher.matches()) {
        // Run undo file
        String runUndoFileCommand = matcher.group(1);
        setThcLteVerdict(moshellSourceEnb.verifyNoErrors());
        setThcLteVerdict(moshellSourceEnb.verifyNoErrors());
        runUndoFileCommand.length() - 4) + ".log")
        setThcLteVerdict(moshellSourceEnb.verifyNoErrors());
    } else {
        setThcLteVerdictAndPrintToLog(new LteVerdict(ERROR,
    
}
"Test case tear down phase went wrong. Could not find the undo file command. " +
    "Searched in string " + undoFileLine));

/**
 * Returns the first line containing the given string from a multi lined text.
 * @param completeText The multi lined text
 * @param stringToFindInLine The wanted string to find line from
 * @return The line containing the string or null if the string was not found in any line.
 */
private String getLineContainingString(String completeText, String stringToFindInLine)
{
    String[] lines = completeText.split("\n");
    for (String line : lines)
    {
        if (line.contains(stringToFindInLine))
        {
            return line;
        }
    }
    return null;
}

/**
 * Returns all lines containing the given string from a multi lined text.
 * @param completeText The multi lined text
 * @param stringToSearchFor The wanted string to find lines from
 * @return The lines containing the string or null if the string was not found in any line.
 */
private Collection<String> getAllLinesContainingString(String completeText, String stringToSearchFor)
{
    String[] lines = completeText.split("\n");
    Collection<String> result = new ArrayList<String>();
    for (String line : lines)
    {
        if (line.contains(stringToSearchFor))
        {
            result.add(line);
        }
    }
    return result;
}
8.6 Java file created by Automatic Test Builder

Automatic Test Builder generates a java file containing following code.

```java
if (result.isEmpty())
{
    return null;
}
return result;

private class trace
{ }
```

---

Package com.ericsson.ate.lte_ran_iov.testcode.generic.automatic_code_generator_hardware_files;

//***Common

import com.ericsson.ate.lte_ran_iov.testcode.generic.automatic_code_generator_hardware_files.ACG_TestCaseMethods;
import com.ericsson.thc.tes.jtex.Verdict;
import com.ericsson.ate.lte_ran_iov.testcase.LteBaseTestCase;
import com.ericsson.ate.lte_ran_iov.testsupport.prepostcheck.PrePostCheckHandler;
import com.ericsson.ate.lte_ran_iov.testsupport.tool.Moshell;
import com.ericsson.ate.lte_ran_iov.testsupport.handler.HandlerManagedObject;
import java.util.ArrayList;
/**
--TC HEADER--
--CTC ID--
LTE00000550;LTE00011137;LTE00074805;LTE00001440
--TC ID--
LTE00000514
--TC Heading--
INT_RANSF024 Add sector, create MOs with moshell
--TC Details--
***/

DESCRIPTION
Integration of use case, feature not fully delivered.
Iteration 2: OaM, RAC, NC

PRECONDITION
AntennaUnitGroup MO and required parents created

TEST INSTRUCTION
Test tool needed for execution:

moshell

TEST SEQUENCE
Create MOs according to MOM in LSV with moshell or AMOS

Define max number of MOs with max number of relations

A limitation in It2 is 1:1 relation between sectorFunction and antennaUnitGroup
The final implementation is 1:4.

Check that all MOs and attributes are correctly created
Create a CV and set the CV to startable
Restart the node
Verify that all created MOs remain and that all attributes have correct values

POSTCONDITION
An executing CV with MOs according to test case is set to startable

PASS CRITERIA
Verify that all created MOs remain and that all attributes have correct values

*******************************************************************************************************
--System Function Group--
*******************************************************************************************************
**** LTE00000550 SFG****
Sector & Carrier Group Handling - RBS
**** LTE00011137 SFG****
Start and Restart - RBS
**** LTE00074805 SFG****
Support System - RBS
**** LTE00001440 SFG****
SW Management - RBS

*******************************************************************************************************
--CTC Heading--
*******************************************************************************************************
**** LTE00000550 HeadLine****
INT_RANSF024 Add sector, create MOs with moshell - RAN_I&V_1
**** LTE00011137 Headline****
RANUC018.A1 Historical oscillator data exist, using NTP clock reference, DUL_Step2 - I&V_Standalone
WP 1477 EXTALM: Update alarm port operational state when RRU contact is established - I&V_Standalone

Basic LTE Support SMO, restore of CV - RAN_I&V_13

CTC SPECIFIC PRECONDITION
Integration of use case, feature not fully delivered. Iteration 2: OaM, RAC, NC

CTC SPECIFIC TEST INSTRUCTION

CTC SPECIFIC POSTCONDITION

PASS CRITERIA

CTC SPECIFIC PRECONDITION

CTC SPECIFIC TEST INSTRUCTION

CTC SPECIFIC POSTCONDITION

PASS CRITERIA
PASS CRITERIA

**************************************
--Quality Level--
**************************************
**** LTE0000550 Quality Level****
N/A
**** LTE0001137 Quality Level****
N/A
**** LTE00074805 Quality Level****
QL3
**** LTE00001440 Quality Level****
N/A

**************************************
--Configuration Values--
**************************************
**** LTE0000550 Configuration Values****
1 Digital Units N/A
**** LTE0001137 Configuration Values****
1 Digital Units N/A
1-12 Radio Units N/A
**** LTE00074805 Configuration Values****
R1 ConfigurationVersion N/A
STP4 ConfigurationType N/A
**** LTE00001440 Configuration Values****
1 Digital Units N/A

*@author eabbsaa
*@time stamp 2012/05/04 18:29:17

----------------------------------------------------------------------------------------
*/
public class testWP1_TestCase extends LteBaseTestCase {
    
    ACG_TestCaseMethods tcMethodsObj=new ACG_TestCaseMethods();
    /**
     * WP1
     **/
    public static enum CtcId {
        LTE0000550, LTE0001137, LTE00074805, LTE00001440
    }
    // #################################################################################################
    // ## MEMBER VARIABLES
    // #################################################################################################
    //###Common

private int m_cellId;
private Moshell m_moshell;
private HandlerManagedObject m_hmo;
private PrePostCheckHandler m_prePostCheckHandler;

/** Sector & Carrier Group Handling - RBS */
/** Start and Restart - RBS */
/** Support System - RBS */
/** SW Management - RBS */

@Override protected Verdict preamble() {
    setTestStep("preamble");
    //***Commo
    tcMethodsObj.init();

    //***Sector & Carrier Group Handling - RBS */
    //***Start and Restart - RBS */
    //***Support System - RBS */
    //***SW Management - RBS */

    return getThcVerdict();
}

@Override protected Verdict execution() throws Throwable {
    setTestStep("execution start");
    //if(isTmRunning("LTE00000550")
    executeLTE00000550();
    //if(isTmRunning("LTE00011137")
    executeLTE00011137();
//iff(isTmRunning("LTE00074805")
executeLTE00074805();
//iff(isTmRunning("LTE00001440")
executeLTE00001440();

return getThcVerdict();
}

// ############################################
// ##              POSTAMBLE                         ##
// ############################################

@Override
protected Verdict postamble() {
   setTestStep("postamble");

   //***Common

   //***Sector & Carrier Group Handling - RBS
   //***Start and Restart - RBS
   //***Support System - RBS
   //***SW Management - RBS

   return getThcVerdict();
}

// ############################################
// ##              TEST METHODS                       ##
// ############################################

private void executeLTE00000550() throws Throwable {
   setTestStep("executeLTE00000550");
   // TODO
}

private void executeLTE00011137() throws Throwable {
   setTestStep("executeLTE00011137");
   // TODO
}

private void executeLTE00074805() throws Throwable {
   setTestStep("executeLTE00074805");
   // TODO
}
private void executeLTE00001440() throws Throwable {
    setTestStep("executeLTE00001440");
    // TODO
}
## 8.7 Time plan

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Task Name</th>
<th>Sub Tasks</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>7,8</td>
<td>Meetings</td>
<td></td>
<td>Meetings with Sören Andersson and Jonas Widén to understand what the problem statement is. Discussing solutions, selecting best possible solution</td>
<td>Completed</td>
</tr>
<tr>
<td>9,10</td>
<td>Learning</td>
<td></td>
<td>Study different reference materials to know how to do the required job efficiently. Learning how to use VE Window Builder. Studying SOAP, Tomcat application server, axis. Learning how to use Rational ClearQuest.</td>
<td>Completed</td>
</tr>
<tr>
<td>10</td>
<td>Environment setting</td>
<td></td>
<td>Setting up Eclipse IDE for Java Developers Helios SR2. Adding required JAR files. Installation of VE Window Builder Studying Mysql. Getting access to the required information. Getting required access rights.</td>
<td>Completed</td>
</tr>
<tr>
<td>11,12</td>
<td>Semi-automated process</td>
<td>Database Connectivity</td>
<td>Meeting with concerned persons and try to get access to the Database TM4LTE</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>Semi-automated process</td>
<td>Database Connectivity</td>
<td>Generating Text file manually</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>Semi-automated process</td>
<td>Database Connectivity</td>
<td>Reading and saving query results from text file to local database</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>Semi-automated process</td>
<td>Database Connectivity</td>
<td>Generating java file automatically</td>
<td>Completed</td>
</tr>
<tr>
<td>12,13</td>
<td>Fully automated process</td>
<td>Connect TM4LTE using SOAP</td>
<td>Designing and writing code to connect database TM4LTE directly from eclipse and fetching data from it. Generating java file and embedding data into java file according to the code template provided.</td>
<td>Completed</td>
</tr>
<tr>
<td>Date</td>
<td>Task Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14,15</td>
<td><strong>GUI design</strong>&lt;br&gt;Mockup development&lt;br&gt;GUI development</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15,16,17</td>
<td><strong>Working GUI</strong>&lt;br&gt;Add real code to GUI&lt;br&gt;Presentation to Mats Wärme, Jonas Widén and Sören Andersson</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Edit code according to the GUI structure to make GUI working.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18,19</td>
<td><strong>Adding main methods and help methods</strong>&lt;br&gt;Designing and creating local database&lt;br&gt;Meetings&lt;br&gt;Specification of all main methods and help methods&lt;br&gt;Adding methods to code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20,21</td>
<td><strong>Documentation development</strong>&lt;br&gt;Writing manual for the Automatic code generator tool&lt;br&gt;Writing Javadocs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td><strong>Training of testers</strong>&lt;br&gt;Delivering the tool to testers&lt;br&gt;Training sessions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23,24,25,26</td>
<td><strong>Writing thesis report</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>