WCDMA Test Automation Workflow Analysis and Implementation

YIKE LIU

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:6

WCDMA Test Automation Workflow Analysis and Implementation

Yike Liu
yi_liu33@hotmail.com

23 April 2009

Industrial Supervisor
Peder Sandelin
Ericsson WRBS I&V

Examiner and Academic Supervisor
Prof. Gerald Q. Maguire Jr.

Department of Communication Systems
School of Information and Communication Technology
Royal Institute of Technology
Stockholm, Sweden

Abstract

In the modern wireless communication industry, radio communication equipment vendors not only produce communication hardware, but also produce software. In fact, software revenue is now a large part of the total revenue. As technology has developed and traffic demands increase, more and more functions required to implement the radio system are implemented via software rather than hardware. Today, many hardware functions are actually implemented with reconfigurable and programmable hardware. Therefore, it is often possible to perform an upgrade by loading new software (a software upgrade) rather than needing to change the physical hardware with every technology advance. However, introducing new elements and features in existing (often mature) software may cause unexpected problems. These problems may include new parts malfunctioning and failure or degradation of old functions. To avoid these problems, each version of software has to be thoroughly tested, not only to test the new parts, but also to verify that the old functions still work properly. Testing all the old functions is time and human resource consuming. Thus, there is an increasing demand for automated testing. This thesis will focus on why automated regression testing is necessary and how to implement automated testing in a specific environment.

The thesis results show that automated testing can improve the test coverage by at least 40% for one of Ericsson's WCDMA software releases. This coupled with a reduction in testing time enables more rapid development by significantly reducing the test time without compromising quality. All of these results lead to improved profitability and increased customer satisfaction.

Sammanfattning

I den moderna trådlösa kommunikationen industrin, radioutrustning leverantörerna inte bara producera kommunikation hårdvara, utan också producera mjukvara. Faktum är programvara inkomster är nu en stor del av de totala inkomsterna. Eftersom tekniken har utvecklat och trafik krav ökar, fler och fler funktioner som krävs att genomföra radiosystem genomförs via mjukvara istället för maskinvara. Många hårdvara fungerar faktiskt genomförs med omkonfigurerbara och programmerbar hårdvara. Därför är det ofta möjligt att utföra en uppgradering av lastning ny programvara (en mjukvaruversionen) snarare än behöver för att ändra den fysiska hårdvaran med varje teknik förväg. Men att införa nya element och funktioner i befintliga (ofta äldre) programvara kan orsaka oväntade problem. Dessa problem kan innehålla nya delar brister och fel eller försämring av gamla funktioner. För att undvika dessa problem, varje version av programvaran måste testas, inte bara att testa de nya delarna, men även för att kontrollera att de gamla funktionerna fortfarande arbete ordentligt. Testa alla gamla funktioner konsumera tid och personal. Således finns det en ökad efterfrågan på automatiserade tester. Den här avhandlingen kommer att fokusera på varför automatiserad regression testning krävs och hur man genomföra automatiserade tester i en viss miljö.

Avhandlingen visar att automatiserade tester kan förbättra testbunt täckning med minst 40% för ett av Ericssons WCDMA programversionerna. Detta i kombination med en minskning av provning tid möjliggör en snabbare utveckling av avsevärt minska test tid utan att kompromissa med kvaliteten. Alla dessa resultat leda till bättre lönsamhet och ökat kundvärde belåtenhet.

Table of Contents

List of Figures

Acronyms and abbreviations

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
ASC	Antenna System Controller
ATM	Asynchronous Transfer Mode
CDMA	Code Division Multiple Access
CN	Core Network
CS	Circuit Switch
Chan_STM1_E1	Channel for STM1 and E1
DRH	Dedicate Radio link Handling
E1	Euro Data Transmission Standard 1 (2.048 Mbps)
E3	Euro Data Transmission Standard 1 (34.368 Mbps)
EDCH	Enhanced Dedicate Channel
EUL	Enhanced Uplink
FCCU	Function Cross Coupling Unit
FVS	Function Verification Specification
GGSN	Gateway GPRS Serving Support Node
GMSC	Gateway Mobile Service Switching Controller
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
HARQ	Hybrid Auto Repeat Request
HLR	Home Location Register
HS	High Speed (short for HSDPA in this report)
HSDPA	High Speed Downlink Packet Access
HSPA	High Speed Packet Access
HSUPA	High Speed Uplink Packet Access
I&V	Integration & Verification
IS-95	Interim Standard 95
ISO	International Standardization Organization
IEC	International Electro Technical Commission
IT	Information Technology
LED	Light Emitting Diode
LTE	Long term Evolution
MBMS	Multimedia Broadcast Multicast Service
MDL	Mobile Data Logger
MIMO	Multi Input Multi Output
MS	Mobile Station
MSC	Mobile Service Switching Controller
MUE	Multi User Equipment
Mbps	Megabits per second
NBAP	Node B Application Part
NITE	Node B Integrated Test Environment

NodeB	3GPP name for a WCDMA base station
O&M/OAM	Operation & Maintenance
QoS	Quality of Service
R99	3GPP UMTS Specification release 99
RAB	Radio Access Bearer
RAN	Radio Access Network
RBS	Radio Base Station
RDBTH	Radio Data Bulk Tracing Handler
RLS	Radio Link Set
RNC	Radio Network Controller
RRM	Radio Resource Management
SGSN	Serving GPRS Support Node
SRB	Signaling Radio Bearer
STM1	Synchronous Transport Module-1 (155.52 Mbps)
SW	Software
T1	US Data Transmission Standard 1 (1.544 Mbps)
T3	US Data Transmission Standard 1 (44.736 Mbps)
TC	Test Case
TFR	Transmit Format Reconfiguration
TMAP	Test Management Approach
TV	Television
TX_DIV	Transmit (Tx) Diversity
TcData	Testcase Data
UE	User Equipment
UMTS	Universal Mobile Telecommunication System
UTRAN	UMTS Terrestrial Radio Access Network
Uu	3GPP defined Air interface (between UE & UTRAN)
VLR	Visitor Location Register
WBTS	WCDMA Base Transceiver Station
WCDMA	Wideband Code Division Multiple Access
WRBS	WCDMA Radio Base Station

Chapter 1. WCDMA Overview

1.1 WCDMA Introduction

Wideband Code Division Multiple Access (WCDMA) is a modern standard for mobile telecommunication systems. More specifically, WCDMA is a third generation wide area cellular communication technology. Unlike the second generation cellular communication systems such as GSM and CDMA IS-95, WCDMA provides better quality of service and higher throughput.

The WCDMA specification was first released in 1999 by the 3rd Generation Partnership Project (3GPP) [9] as release 99 (R99). WCDMA was designed for multimedia communication. More specifically it was design to provide the user with video calls, internet access, and video streaming so that users can utilize multimedia applications, such as Mobile TV. The higher data rates provided in WCDMA enables a content provider to provide higher quality image and video services than the earlier GSM system. The system not only provides a more flexible and high quality service infrastructure, but also creates a large market opportunity for application and information (e.g., content) companies.

As the technology developed and live networks were launched, R99 could no longer satisfy users, operators, or the cellular equipment vendors. Thus, a technology evolution has occurred. The most important features that have been proposed and implemented are: high speed downlink packet access (HSDPA) and enhanced uplink (EUL). By optimizing the R99 time and code resources, WCDMA has evolved to provide downlink data rates of up to 14.4 Mbps.

Today, a new evolutionary step, Long term Evolution (LTE), has been proposed in 3GPP. LTE is expected to provide up to 100 Mbps throughput. It is believed by some that in a near future, wide area cellular systems will provide such good service that it will be possible to mobilize all office work, i.e., there will not longer be a need for fixed telephones or fixed office locations.

All of the evolutionary steps mentioned above could be realized with very limited hardware modification, as most changes could be made via changes to the basestation and core network software.

1.2 WCDMA Architecture

This section introduces the basic concepts of a WCDMA system by presenting its architecture, including the key components and the interfaces between all the network elements. The WCDMA architecture is similar to the 2nd generation GSM cellular system. This was done purposely by 3GPP to enable 2nd generation systems to be

upgraded in steps to a 3rd generation system while minimizing the cost of such a transition.

The WCDMA system can be divided into two parts: Universal Mobile Telecommunication system (UMTS) Terrestrial Radio Access Network (UTRAN) and Core Network (CN). UTRAN provides the radio functionality, such as wireless data transmission, call service, and so on; while the CN is responsible for of all the switching and routing of service requests and data – along with interfacing to external networks. The basic architecture is shown in Figure 1. As shown in this figure, the interface between the User Equipment (UE), also called the mobile station (MS), and the UTRAN is called the Uu interface, while Iu is the interface between the UTRAN and the CN.

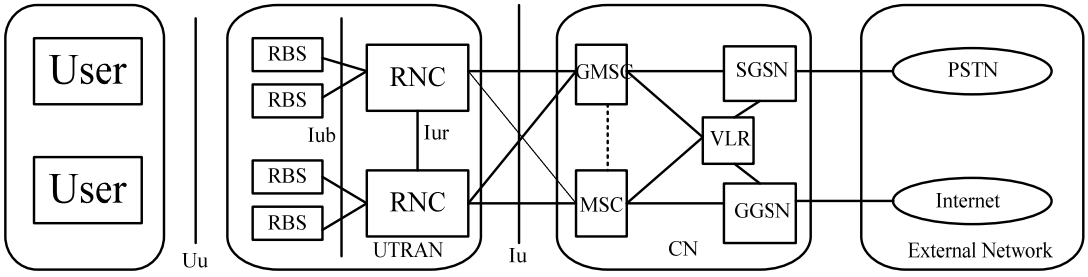

Figure 1: WCDMA architecture overview

1.2.1 UTRAN architecture introduction

UTRAN consists of two parts: Radio Network Controller (RNC) and Radio Base Station (RBS). The RNC controls all the radio resources in its domain, including all the physical resources such as the RBSs connected to it and all of their the logical resources (for instance the code resources and interference resources). The RBS converts all the data received from the RNC on the downlink and all the data from the UE for the uplink into/from encoded and modulated radio signals. The RBSs converts all of the downlink data to a radio signal and sends it to one or more UEs. Similarly the RBS converts the received uplink radio signal from the UEs into data and sends it to the RNC. However, as part of the HSPA evolution, the RBS performs some of the Radio Resource Management (RRM) that in earlier releases was handled by the RNC. Specifically, some of the RNC functions such as scheduling have been moved to RBS, in part to reduce retransmission delay. Figure 2 illustrates the UTRAN concept.

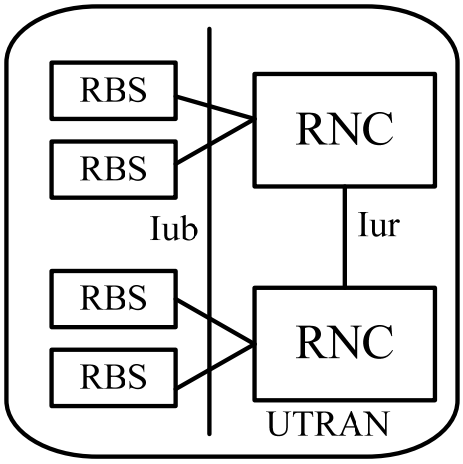

Figure 2: WCDMA RAN architecture

1.2.2 CN architecture introduction

The Core Network (CN) of an existing GSM network can largely be reused for a WCDMA network. As mentioned above, this provides compatibility and cost reduction. The CN includes the following subsystems:

Home Location Register (HLR): The HLR is a database located in the subscriber's home system that stores the master copy of the subscriber's service profile. When a user subscribes to a network, the HLR is updated to include the basic information concerning this subscription. The HLR also stores the subscription's Quality of Service (QoS), as well as (current) roaming information and location information.

Mobile Service Switching Controller (MSC) / Visitor Location Register (VLR): The MSC is a circuit switch and the VLR is a database. Together with the RNC and RBS they provide Circuit Switched (CS) service to the UE in its current location. The VLR stores all the visiting subscriber's service profiles.

Gateway Mobile Service Switching Controller (GMSC): A GMSC is a switching point between the WCDMA system and external CS networks. The GMSC acts as a circuit switched gateway, hence all incoming and outgoing circuit switched traffic passes through the GMSC.

Serving GPRS (General Packet Radio Service) Support Node (SGSN): The SGSN is similar to a MSC/VLR, but for packet switch service. More specifically it is a router for packet traffic.

Gateway GPRS Serving Support Node (GGSN): the GGSN provides for the packet switched domain similar functions as the GMSC provides for the CS domain.

The CN architecture in WCDMA is shown in Figure 3. Further details of the WCDMA architecture can be found in [10]. Additional information about HSPA and LTE can be found in [11].

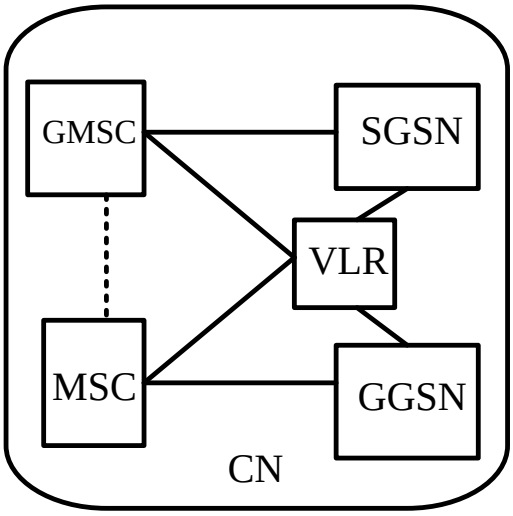

Figure 3: WCDMA core network overview

Chapter 2: Automated Testing

2.1 What is software testing?

2.1.1 Definition of a test

According to the International Standardization Organization (ISO), a test can be defined as "Technical operation that consists of the determination of one or more characteristics of a given product, process or service according to a specified procedure" [22]. Testing examines **if** a certain item meets specific requirements. Generally this testing is done to ensure that a certain item meets a commonly used set of requirements (where these requirements are often specified by a standard or by a standardized testing procedure).

In the IT industry, especially the software industry, testing is usually performed to ensure that the product has all the required features and has no bugs. This requires testing all the (new) features which have been implemented in a new build of the software, along with sufficient test coverage to ensure that all of the old functions continue to work correctly. The later form of testing is referred to as *regression testing*. Regression testing is designed to ensure that the introduction of new features or improvements to old features does not introduce bugs into the software.

2.1.2 Why testing should be automated

As stated in the previous section, test coverage must be sufficient to detect errors in the device under test. In backward compatibility test, the purpose of the test is to verify that newly introduced software is compatible with the previous version. Thus for each new version of the software, it is important to determine if this new version will degrade the system's performance (in comparison with the performance of the earlier version of software). There is generally pressure to both improve a system's performance and to implement new features. However, it is important to keep the system stable – i.e., so that it continues to correctly perform the tasks that the previous version of the software correctly performed. Regression testing has to be performed for each new version of software. This regression testing can be very time consuming because the test must cover all features of all prior versions (or in most cases at least the previous version) and verify the stability of the new release. Additionally, testing is needed to verify that the new version correctly implements the new features and functions. As the complexity of the software for the systems of concern to this thesis, a very large number of tests are needed. Therefore, it is highly desirable to implement a tool to automate testing (especially regression testing) as this will save a lot of human resources and reduce the cost of the verification process.

The following are some of the advantages of automated testing:

- Automated testing will save time during the verification process
- If the testbed or test platform is designed to be a general testing platform, then it will accommodate many different test cases. This increases the number and variety of tests which can be automatically performed – reducing personnel costs and reducing testing time.
- Human resources and costs can be reduced by using an automated test tool.
- Reducing the testing time can reduce the software verification life cycle (i.e., detecting errors sooner can enable rework or correction sooner – which means that the product is able to be shipped sooner and hopefully reduces the need for field maintenance).
- Eliminating the need for human interaction during the verification process can reduce testing cost and decrease testing time; additionally, it permits the testing to take place at any time of the day or night.
- Automated testing permits better test coverage for each software build (as some of the time saved by automating the testing can be used to run a wider set of test cases).
- Automated testing can help reduce the time need to identify potential bugs introduced by new features; thus enabling them to be corrected sooner rather than later.

However, even with all of the advantages above, automated testing has its limitations. Some of the drawbacks are:

- Maintenance is required. Automated test tools need to be supported by a group of talented engineers who are familiar with both software engineering and the specific system being tested (in this case, a basestation of a WCDMA system). A group is responsible for developing the automated testing tool and will maintain the tool(s), or another group will be required to maintain the test tools. The expense of developing automated testing and the resulting cost savings must be calculated and compared to manual regression testing.
- Poorly designed test cases will reduce the performance of an automated testing tool. (Just as poorly designed test cases reduce the performance of manual regression testing.)
- New bugs can be introduced by each new version of the automated testing tools. Especially damaging are bugs due to added facilities that were inserted into the software in order to perform or facilitate automated testing.
- Automated testing tools require a very stable test environment. This is because if is very hard to tell if the reason a test failed was because the device under test actually failed or if the test environment is incorrectly configured or operating incorrectly. Thus, if there are failures detected by an automated test,

then human testers have to identify whether the failure was caused by the test tool, the test environment, or the actual test target.

However, it is widely believed that automated testing can significantly improve the verification process and increase the performance of the testing process. Today more and more IT companies are adopting automated testing. Therefore, there are lots of employment opportunities within this area – while the number of manual testers decreases.

2.2 Software Test Basic Concepts

2.2.1 Test Organization

For both system and software testing, a good test plan and organization will dramatically improve test efficiency. The basic concepts and test organization design have to be clear and the tests well designed in order to achieve stability and good test coverage.

The following points are the essential factors in the test process:

- Test Cases

  Software and systems will be run in different scenarios such as different radio access bearers (RAB) establishing requests, different transmission types of requests, etc. Most of the bugs are introduce during implementation. Usually these bugs are due to improper implementation of algorithms [1]. Test cases are designed to simulate a variety of different scenarios. Moreover, a specific test case is usually designed for each new software feature to establish that the new feature or function works. The test cases also include some regression cases to ensure functional compatibility of the software. For instance, Ericsson's latest WCDMA base station software introduced a feature called *enhanced uplink* with a 2 millisecond transmission interval. The test case to test this feature requires HSDPA service, since the enhanced uplink service has to work together with HSDPA. Thus, this test case must not only test that the feature works, but also test that the two features (enhanced uplink and HSDPA) are compatible with each other.

- Test Object

  In this thesis, the term **Test object** means the test target. This target can be a specific function or a specific combination of functions of the system being tested (often called the "system under test") that are to be tested. For example, the hardware test object includes a hardware test for each connector of the device under test (often called the "unit under test").

- Test Coverage

  Test coverage refers to the fraction of the entire program that is tested by the test cases.

As mentioned in previous section, software developers keep adding new features, so that users will want/buy upgrades. This is one of the most common sources of problems in the IT industry. However, each new version or build of software will include some new functions that the customer requires and will often contain a large number of features which the customer does not require, but that are there for compatibility with a previous release, for test purposes, to meet legal requirements, etc. Therefore, the quality of the software is not only a function of the new features, but will depend upon how well all of the functions work. Thus, testing of a new build has to cover both the new functions and the old functions – in order to ensure that the system works correctly after a software upgrade. Since there are so many functions in many systems, it is not possible to do an exhaustive test of all the functions for all inputs and under all conditions. Therefore, test coverage definition is a very important element in the design of the test process.

In the IT industry, many companies use IBM's Rational Tools to design test cases [12]. Since today most software development is modularized, hence the test cases, especially for black box testing, are designed to test the modules in combination. [4] This assumes that element tests have already been performed when each module was being developed. For this reason the type of testing that this thesis is concerned with is often called "integration testing".

- Test Suite

  A **test suite** is a series of test cases. A test suite usually is the main product of automated testing development. This test suite includes all the test cases that have been automated. These tests can subsequently be executed for each software build.

- Regression Test

  In this report, our focus is the design of regression testing. Since test coverage has to be expanded to cover all the new functions and the old functions, the regression test must include legacy tests (i.e., to verify the proper functioning of all the earlier functions). To prevent the test suite from continuously growing requires either developing test cases that provide improved coverage with a smaller number of test cases (using tests that combine the effects of multiple separate tests) or eliminating some of the tests of earlier functionality. To facilitate the later, the regression function verification specification may allow some very old function test cases to be removed from the test requirements.

- Black Box

  Software testing can be divided into two types: white box and black box. White box testing is primarily done at the time of development – as it requires access to the source code, while black box testing is done for

acceptance testing [6]. As acceptance testing in the focus of this thesis, the testers do not use the source code to the WCDMA Radio Base Station (WRBS) software, thus all the test cases are performed in black box manner. We will assume that earlier white box testing was done to ensure that each subsystem has been tested in isolation. Black box testing is necessary because the focus of the acceptance (and integration) testing is verification at a system level and it is not feasible to test every subsystem's implementation. Therefore, we are concerned only with the output of the test cases, i.e., that the output of the test case matches the expected output.

- Test Environment

  The test environment includes all the test software and hardware that are require for the test, such as all the simulators (to provide input), test instruments to observe the output of the equipment under test, and the different configurations of the test object.

When automating the test process, all of the above points have to be addressed. A well designed test process will improve the test efficiency. For instance, in Ericsson's WRBS I&V (Integration and Verification) department test analysis is performed before the test process starts. Based on test procedure analysis and test analysis 30% of the regression test cases can be eliminated [3]; both reducing the duration of testing and making it easier to identify bugs. However, if the test process is not defined properly, it can introduce additional problems during the test. [6]

2.3 Ericsson's WRBS Automation Introduction

Ericsson's WRBS I&V (Integration and Verification) department is currently implementing an automated software test tool for regression testing. The goal of the project is to automate all the test cases that do **not** need human intervention. The reason for this development is that Ericsson's WRBS includes so many features that it is almost impossible to test all the previous functionality using human testers. Thus, to expand the test coverage and reduce labor costs, use of an automated test tool is proposed. This thesis project was conducted as part of this effort.

2.3.1 Ericsson WRBS I&V Test Target

In chapter 1, the basics of a WCDMA system were presented. The RBS acts as a translator or relay node in UTRAN – as it receives a radio signal from a mobile station and transfers the signals and data to RNC via a transmission network. Due to the evolution of WCDMA networks (such as the introduction of HSPA); the RBS has become responsible for resource allocation in the local cell. Moreover, the deployment of Hybrid Auto Repeat Request (HARQ) in the RBS requires that the RBS take responsibility for physic layer retransmission in order to reduce delay.

The desire is to verify all the features (including hardware control functions and data processing functions - so called traffic functions) that have been implemented in a new build and to ensure the stability of the software running on Ericsson's

WCDMA RBS. Thus a WCDMA RBS is the test target for all of the activities described in this thesis.

2.3.2 Ericsson WRBS I&V Test Platform

Ericsson WRBS I&V department tests all the WCDMA RBS software for a specific platform - including all the hardware and software.

- Hardware platform

  The WCDMA UTRAN consists of three parts: RNC, RBS, and MS. For testing purposes a RNC simulator and MS simulator are used to ensure a stable verification environment. Thus during the test process, a fixed RBS hardware platform is used – therefore the RBS software is the only variable element during verification testing. Eliminating hardware variability should make it much easier for the testers to identify bugs in a specific build of the RBS software. Note that this requires testing to verify the hardware platform's correct operation, before it is used for software testing.

- Software platform

  Since the RNC simulator and MS simulator provide open Application Programming Interfaces (APIs) and each has a command line interface, Ericsson's WRBS I&V unit has developed a software platform called the NodeB Integrated Test Environment (NITE) to control the three parts during a test, i.e., to control the RNC simulator, the RBS, and the MS simulator in the test environment.

2.3.3 Automated Testing Tool Development

In this thesis project, an automated testing tool is developed based on the software and hardware platforms mentioned above. NITE provides the ability to control basic operational functions of the RNC (simulator), RBS, and MS (simulator). Using NITE's open APIs, testers can write scripts to automatically perform each step in the test process. The NITE platform is implemented in Python[1], thus, the automated test scripts will also be written in Python.

Usually, a software build test includes many test cases. In our automated test tool, each test case will be a Python script. The automated test tool generates a set of scripts that implement each of the required test cases.

[1] More information regarding python can be found at http://www.python.org .

Chapter 3 TMAP in Ericsson

3.1 TMAP introduction

3.1.1 What is TMAP?

This section describes the Test Management Approach (TMAP) [5] that will be used. The test process plays a very important part in software development. In the IT industry, the number of testers is often equal to or greater than the number of developers, in some companies this ratio can even be 1.5 or 2; thus the cost of testing is quite significant.

The reason why software products need so many testers is the company's desire to ensure sufficiently high quality software and the financial risks associated with delivering low quality software. Additionally, the trend toward continually adding new features both adds new code that must be tested and can bring previously hidden bugs to the fore. With a large number of testers and good management of these testers, high test coverage is possible, i.e., the superset of all the test cases written these testers can include more regression test cases. In Ericsson WRBS I&V, the test cases are divided into work packages. Each work package includes not only the new features, but also some regression functions which are related to the new features. However, as there are a large number of legacy functions (that need regression testing) and there are new functions that need new test cases, but the number of testers is limited, the number of testers is usually less than necessary. Adding additional testers, enables each work package to include test cases for new features and to add additional regression test cases to expand the regression test coverage.

Unfortunately due to differences in the skills of testers and the interdependence of functions, increasing the number of tests does not lead to a linear increase in test coverage. Simply increasing the numbers of testers is not sufficient, as poorly designed test processes and poor organization will cause the test process to yield poor results. That is why TMAP is used many companies to utilize their resources more efficiently. These resources can be reduced by utilizing a better management method for software development and testing. An introduction to this better managmement method is presented in the next section.

3.1.2 Basic TMAP Concept

TMAP considers and defines all the activities (including planning, preparation, and execution) that may happen during the test process. This test process is organized as a life cycle. The factors that will influence the lifecycle in a test process are: techniques for testing activities (T), organization embedding (O), resources & infrastructure (I), and the lifecycle process of the testing itself (L). Figure 4 shows the relationships between these 4 cornerstones of TMAP. In the following sections, each

part will be briefly introduced to clarify why TMAP and automation are necessary to support the software development and testing process.

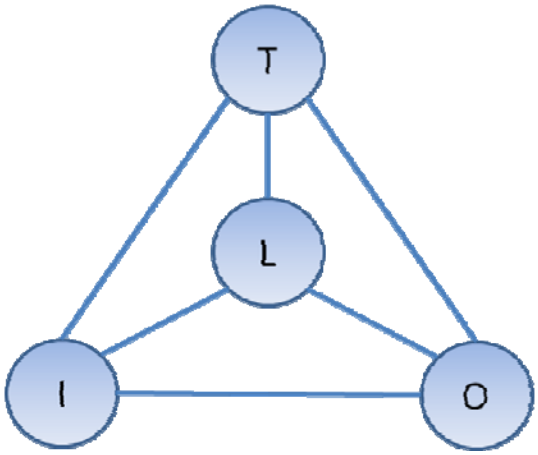

Figure 4: Software lifecycle TMAP model (based on the figure shown on page 35 of [5])

3.2 TMAP Components

3.2.1 Lifecycle model for testing activities

The lifecycle model for testing activities determines how the test process progresses. As a tester, this model describes how the test can be performed, what kind of tools and environment are required, and who will be the responsible for the test process (including allocating the required resources). Each phase in the lifecycle has certain pre-conditions and activities. The following paragraphs briefly describe these phases, more details can be found in [5].

The lifecycle concerns the entire test process. Thus the lifecycle model describes all the factors that might influence the test process, how to prepare for different situations, and most important, the plan for the whole test process. A well designed lifecycle model will drive the test process in the predefined direction.

As mentioned above, planning, preparation, specification execution and completion [5] are part of each and every part of TMAP (lifecycle, technology, organization, and infrastructure). In the lifecycle, the planning part defines a detailed plan for the test process and answers all the questions listed at the beginning of this section; for example, how, with what, under which preconditions, and so on. The lifecycle part of TMAP is the core of the test process. After the test specification has been agreed upon, the lifecycle plan controls the test process.

The main task during the lifecycle planning is to define the test assignments, establish or define the test environment, setup a test control organization, and specify the time plan and schedules. The test control organization usually generates the plan.

However, since some situations may occur that were not considered during the planning, the plan may have to be refined as the test process matures. Thus it is important to recognize that the test process must be adaptive to the actual testing results and business needs, thus the lifecycle plan is a living document and not simply fixed in stone.

3.2.2 Techniques

The techniques that are going to be used for each of the various test activities have to be thoroughly considered when the test process is being *designed*. Since several activities (Planning & Control, Preparation, Specification, Execution, and Completion) happen during the test process, it is necessary to know how the activities can be finished.

Before the test process starts, the techniques in all the activities must be evaluated and the proper testing techniques chosen to satisfy the test requirements; while also offering test efficiency. This is why Erik van Veenendaal and Martin Pol's TMap paper[6] talks about "usable techniques".

During the planning & control phase, the development strategy must be selected and a schedule analysis must be done. Assuming the selection of an excellent technique in this phase, then the test strategy and schedule will be determined based upon the product requirement from customers. Since the expectation from the customers for the product varies depending upon the different scenarios in which the system is going to be used by the customers, thus defining a good test strategy and schedule to meet these various requirements is difficult. It is reasonable to expect that the first test cases will need to be based upon the most common scenarios, while later test cases will cover the special cases necessary to a more limited set of customer requirements (for example, there might even be a branch in the test suite based upon the type of customer or even the risk tolerance of the customer – as some customers may find it advantageous being on the "bleeding edge" while other customers will want to operate a system which is already in widespread use (i.e., late adopters)).

In the preparation phase, the sub goal is to clarify the test bed that is necessary for the test process. In specification phase, test case specifications driven by the test strategy and test bed will be determined by test case analysis. During the execution phase, the testers follow checklists and use statistical methods to determine if the require features work correctly and if the system meets its requirements (including the desired stability). Statistical control models (such as six sigma) are widely used in manufacturing, but are not as widely used in the IT industry (however, there is some work in this area, see for example [23][24][25][26][27][28]).

3.2.3 Organization

The complete test process is a complicated process which involves many factors including hardware, software, and human beings. It is both desirable and necessary to

assemble all the relevant resources into a highly efficiency testing system. To construct a smoothly working team, requires that all of the following be considered:

- Who will be the testers?
- Who will be the team leader?
- Who will be the hardware management and support team?
- Who will perform the test monitoring?
- Who will be the test case designers?
- Who will be members of the control committee?
- Who will coordinate the various teams? What communications and technology will be used for this coordination?
- Who will be the automation architect?
- Who will be the automation engineer?

Considering all the different roles above, it will be difficult to find a set of persons with the required competences. Thus, when assembling the team, it is very important to choose the proper persons.

Moreover, communication is also important inside the test team. As soon as the test plan has been defined, execution ability must be given the highest priority. At this phase, communication between the management team and testers will be essential; this requires good organization and a suitable set of communication channels.

3.2.4 Infrastructure

The infrastructure for test process includes the test platform and test bed. The test platform includes the hardware and software, and also the test procedures, test environment construction, etc.

The infrastructure should be designed to provide stability and minimize variations. To ensure that the test process can be executed smoothly, the infrastructure has to be constructed to enable the optimal combinations of test inputs to be applied to the system being tested. As the goal of testing is to determine if the product meets specific requirements, a stable test infrastructure can make the test results more convincible and much easier to analyze with statistical methods – since the variation should only be due to the system being tested and not due to *unintended* variations in the test environment. Since testing will be applied to many versions (both now and in the future) it is important that the test environment be stable over a long period of time – in order to decrease the difficulty of analyzing the test results.

Therefore the test infrastructure requires:

- the hardware necessary for carrying out the test suite(s),
- the software necessary for carrying out the test suite(s),

- the procedures for carrying out the test suite(s),
- the necessary test environment to be constructed,
- functioning internal communication between the members of the team,
- the test environment must be stable, and
- Ideally variations should only be due to differences in the operations of the system being tested.

Note that in the above there is reference to the possibility of multiple test suites, this is important to consider as future tests (i.e., of subsequent builds/versions) of the system may require quite different test suites than are used for earlier tests. As noted previously the test process has to be adaptive to these changes and the changing requirements of the company and its customers. Clearly there is a conflict between being adaptive and maintaining stability, thus the test team must understand how to make this trade-off over time.

3.3 Ericsson's WRBS I&V Software Test Process

3.3.1 Ericsson's WRBS Software Test Projects

In Ericsson's WRBS I&V department, a release of software will be handled by different project teams. These teams include an integration team, traffic team, Operation & Maintenance (O&M) team, and legacy maintenance team.

Before a new software build for a WRBS is released, the integration team will test the sub-modules of the software. This integration team is responsible for ensuring that all the sub-modules work and that the basic functions of these modules do not conflict with each other. After the integration team has verified that all the sub-modules work correctly, then these modules will be built into a package of software (called an upgrade software package) and transferred to both the O&M and traffic teams.

Within a telecommunication operator, O&M refers to the equipments' operation and maintenance. The O&M team is responsible for testing the O&M functions of the upgrade software package, such as locking and unlocking boards, correct licensing operations, generation of alarms, and so on.

The traffic team works in parallel with O&M team. While the O&M focus is on the operations and maintenance performance of the test target, the traffic team's focus is on the test cases related to actual (data) transmission such as the Radio Access Bearer (RAB) setup and so on.

After the O&M and Traffic teams' testing are completed, the next focus is on testing the new features and functions of the release. This must be followed by some regression testing. Only after all of these tests have been satisfactorily completed can the product be released and be installed in live networks. Following the product

release, the maintenance team takes responsibility for the upgrade software package (as the completion phase mentioned above). The completion phase keep track of the release software and corrects potential bugs which might occur in a live network. The maintenance team's major job is to ensure that the software is running correctly in live networks and trouble shooting any problems which appear in the field within this product. The maintenance team must provide feedback to both the software developers and the various test teams – as if a problem is detected once the software is installed in the field, then there has been a failure in the design and implementation of both the software itself and *of the testing process*!

3.3.2 Ericsson's WRBS I&V Workflow Model

As the section above described, the work flow within Ericsson's WRBS I&V department is primarily linear, but with feedback paths. Figure 5 illustrates this structure (which can also be seen as a "work flow").

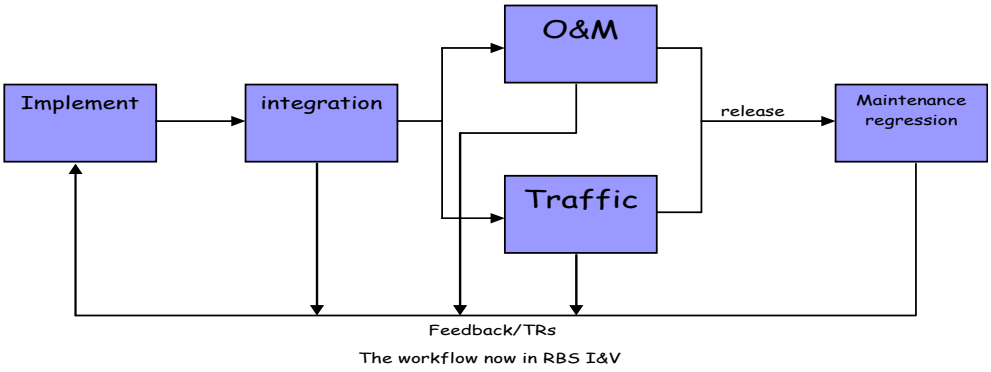

Figure 5: Ericsson's WRBS I&V workflow model

As the figure above shows, the linear structure mainly focuses on the new features, while the legacy tests will mainly be done during the maintenance phase. Unfortunately, this means that backward compatibility testing will mainly occur in the live network. However, if the software does not work in the live network, then the product quality as seen by the customers will be considered poor. This may affect Ericsson's reputation and the willingness of the customer to purchase/license new features – as they will come as part of a new package. Therefore there is a desire to change this workflow.

3.3.3 Automation Test Workflow

With the traditional work flow currently used by Ericsson I&V, even though the regression test cases have been chosen very carefully, there is a high level of risk and a very large human resource requirement to conduct these tests. Additionally, the existing manual regression testing can no satisfy the test coverage requirement. To solve this problem, it is necessary to move regression testing to an earlier phase. However, if the regression testing is not automated, it will be almost impossible to move it, due to the test plan schedule and human resources available. Thus, a decision

was made that an automation test tool was very necessary; not only to reduce testing time and to allow easier execution of these tests, but also to dramatically decrease the risk associated with releasing a software product/package.

Introduction of an automated testing tool should improve the efficiency of this lifecycle - saving a lot of expense in the long run. Figure 6 illustrates the proposed work flow model and shows how regression testing is now integrated into all of the elements of this workflow. Moreover, since regression testing has been moved to (and integrated with) the development phase, this can also reduce the product development time cycle. As discussed earlier, understanding the lifecycle model of the software testing process is essential.

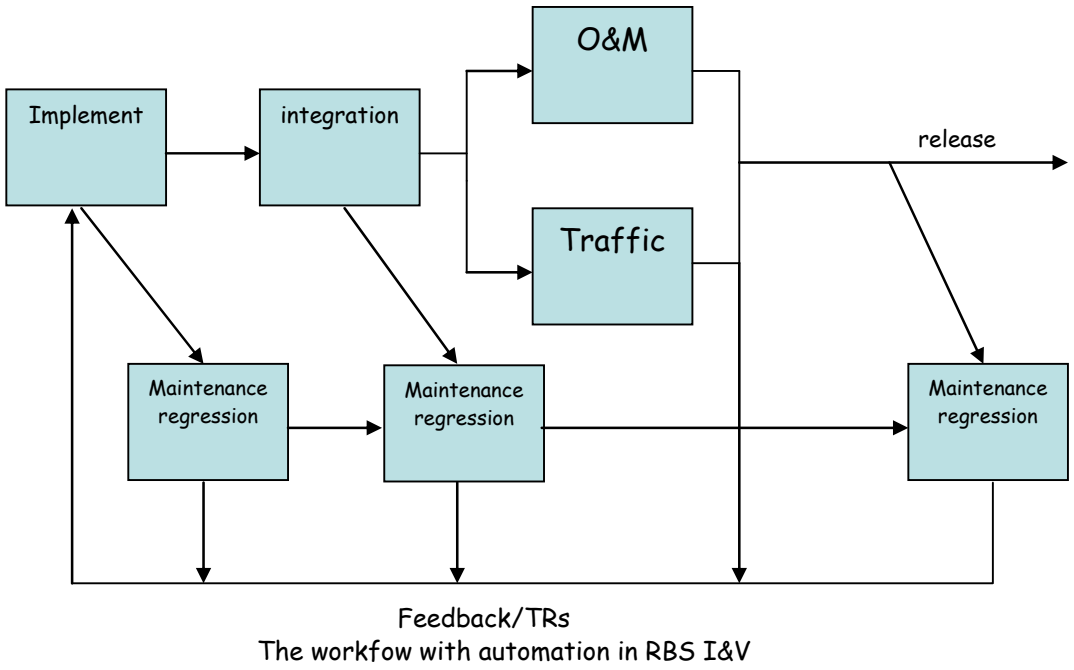

Figure 6: Ericsson WRBS I&V workflow with automation

3.3.4 Benefit of TMAP when using Automated Testing

TMAP is a software test process management approach with four important parts: lifecycle, techniques, infrastructure, and organization. Using an automated testing tool, decreases the critical paths in the lifecycle shortening the time before a product/package can be released, thus accelerating product manufacturing. An automated testing tool offers an efficient way to perform the tests in a test suite. Since the automated testing tool removes the need for human interaction, this testing technique should eliminate many of the human errors of both maintenance and regression testing. Moreover, the automated testing tool will manage the infrastructure, thus reducing the overhead imposed on the organization - while providing a more stable test platform – hence reducing the potential variability. This in turn should mean that deviations from the expected test results reflect the presence of errors or bugs in the software and not errors in the conducting of the tests. Note that

human errors may still occur as the unit under test must still be placed in the test environment, connected to the test equipment (in our case the simulators for the other parts of the WCDMA system and other test instruments). Unfortunately, this testing technique does not eliminate the problems due to errors or failures in these simulators or test instruments. This is the reason that we introduced test cases integration work as the last phase to form a test suite.

When test case scripts have been completed, integrating these scripts into the test suite will be done. The integration work is not only testing the scripts themselves, but also integrating the other components, MSsim, RNCsim, NITE platform, etc. to find a optimal test environment combination. Usually, during the integration work, only one of the varibles (i.e., one component) will be changed. The new test suite will be released only after integration. The combination forms an entire release product including all components' configuration.

Chapter 4 Automation Test Tool Design

4.1 Automated Test Case workflow

4.1.1 Automated test case development workflow

It has been stated earlier that during the software development and test process, the planning and control phase is very important. Thus, to automate the entire regression test, the test cases need to be carefully designed and well organization so that the test script developers will have a clear architecture and logical flow to realize.

After the test cases have been defined, those test cases that can be automated will be selected and prioritized. To ensure product quality, the basic and most important test cases will be given the highest priority. After the test cases have been organized, according the test script workflow, then the test developers will implement these test cases according to the test specifications. These testing scripts will themselves go through a review process. After this review, the test scripts will be passed to the integration testers and used to test a number of different scenarios; such as different RBS configurations, interaction between the test cases, etc. If any problems are detected, then the integration testers will inform both the software developers and the test developers – so that they can both adjust their code to accommodate the various scenarios – thus providing a more general platform for both the software and the test cases. As soon as the integration work has been completed, the test cases will be added to the test suite and released to all the other RBS software testers to use. This work flow is illustrated in the following figure. Note that the automated analysis reuses test analysis, as automated testing replaces the manual regression testing. The only unique part is the automation test cases prioritization part (mentioned at the beginning of this paragraph).

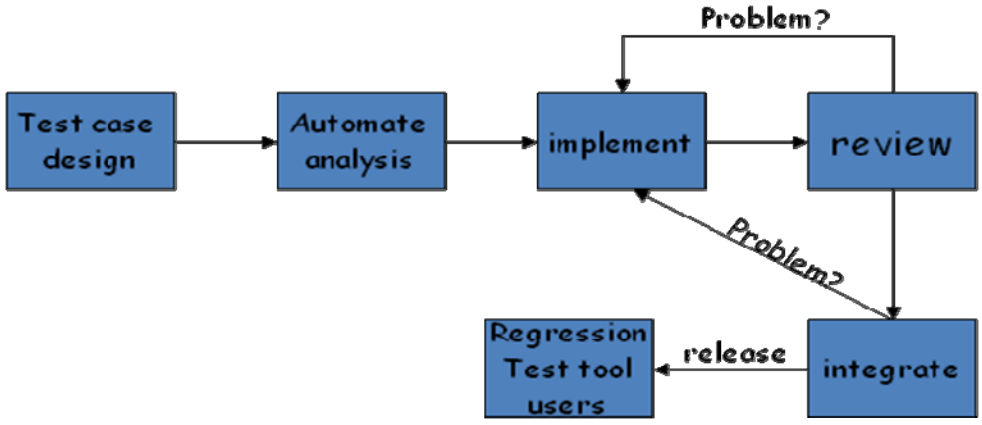

Figure 7: NITE script (automation) development workflow

4.1.2 Automated test case maintenance workflow

During a software package's lifecycle, traditionally more time is spent maintaining the software product than was spent to develop it. However, this is generally a highly inefficient approach – as the cost to correct an error increases (greatly) with the delay between design & implementation and detection. Thus the proposed new workflow integrates maintenance testing with the design and implementation workflow. Obvious benefits are reduced development time, reduced cost of late stage rework, and reduced incidents of bugs being detected after the software package has been released (leading to reduced risk).

As mentioned in the previous section, the test suite will be used by the regression testers. During the test process, these testers will probably find errors caused by both the test target (RBS software) and the test tool. These testers must analyze these failures and identify which type the failure is. If the failure is caused by the test tool, then the testers will submit to the test developers a trouble report stating the error or submit a change request to state the necessary change. If the failure occurs due to an error in the test target, then a trouble report or change request will be submitted to the software developers and designers. These error reports should all be analyzed to improve the test coverage of the tests and as input to statistical process control.

4.2 Automation Test Tool Design Rules

4.2.1 Scripts Automated Precondition

All test scripts intended to run in an automated test suite must require no manual intervention. Thus all the regression test cases can be run during the night. Unfortunately, this means that test cases which require a human tester's monitoring can not be included in this test suite. For example, some of the test cases may require the tester to examine status lights on the RBS, or the test case may require the testers to remove or insert a single board. Such test cases will not be considered automated test case candidates (at least not initially). However, in the future, the department is planning to utilize a web camera to record the lights' status, to enable automation of the test cases which require verifying an LED's status. Note that some of the test cases involving removing and replacing boards can be implemented using computer control of the specific board (for example, by powering it down or setting it into an off or standby state).

4.2.2 Predefined Method Usage

The automated testing tool to implement the legacy regression test suite was developed based on the NITE platform. This platform provides some predefined functions to control the basic operation of the RBS, such as cell setup, radio link setup, etc. The names of all of these functions have the prefix "AL_API", indicating that this function is provided by NITE platform. When developing a test script, it is

very important for the developers to use the standard functions provided by the test platform to reduce the maintain costs associated with each test script.

The test cases will be divided into several test groups (a sub test suite) based upon the different test targets. For instance, the hardware test cases will be grouped into one sub test suite, while the common channel control cases will form another sub test suite. The test script team is responsible for developing a test base class for each sub test suite. The functions in this class will be used throughout the various test cases within a certain sub suite. This base class will be the super class for all test scripts included in a sub test suite – thus a sub test suit is itself an object. The common functions of this base class will be imported before any test case runs. After the base class has been designed and implemented, the test cases can be coded to use the functions of this base class. Details of this base class can be found in [7].

However, NITE and this base class may not provide all the functions needed by a test script developer. Therefore test script developers will have to implement their own functions within their script. If these functions can be used by another test case in the future, then these functions should be moved to the test base class or to a base class for a specific group of tests (i.e., to the sub test suite).

4.2.3 Comments in the Code

In software engineering development is only a part of the whole process. As noted previous, traditionally a large amount of the total effort is spent on software maintenance. Moreover, as the human resources assigned to a product change, the software development and maintenance tasks must be handed over to new people. Therefore, suitable documentation is very important so that a new engineer can understand the code, modify it, and maintain it. Comments in the code itself are one means to provide this document. Therefore, as part of the documentation procedures a clear and consistent commenting style is important. This style is document in “NITE TestScript Design Rules” [7].

Note that today there exist a number of tools (such as Doxygen [13]) to automatically extract documentation that is included in the source code to produce documentation for the code. Current NITE (NITE provides the documentation generating function) is being used to extract the documentation embedded in the source code.

4.2.4 Logging

Another important utility to develop software is a logging system. It is unlikely that new software or a new script will be without any bugs. To identify the location of errors in the code, logs sometimes offer a way to find the where the problem is. Moreover, the log can also be used to monitor a running software process. In the NITE scripts described in this thesis, the logging system is divided into three layers. The first layer is called the normal layer. The main function of this layer is to monitor the software’s activities. The second layer, also called the warning layer, contains

some additional information to indicate that there *might* be some error. The third layer, the debug layer, contains all the details generated by running the scripts. Developers usually depend on debug logs to maintain the scripts.

Note that logs are particularly important for automatic testing as (1) there is not a human watching the program's output and (2) the logs can automatically be compared to the expected output – thus reducing the amount of information that needs to be presented to the (human) tester to only those log entries that are relevant to a deviation from the expected output.

4.2.5 Result Report

One of the design rules [7] requires that at the end of every test case, the result of this test case should be output to the log [7]. This result can be: Pass, Fail, or Skip. In addition, to this trinary result, the test case identifier, test case name, failure/skip reason (if a fail or skip occurred) will be logged to make it easier for testers to monitor the test process. Each of these outputs is generated when the indicated conditions are true:

FAIL	FAIL is reported if the preconditions are not met or if the test case result is not OK.
SKIP	SKIP is reported if the actual test configuration does not fulfill the required configuration specified in the precondition. If the test case requires a specific configuration not covered by the precondition, then a TestSkip_Exception can be raised.
PASS	PASS is reported only if no Skip or Failed exception occurs.

4.2.6 Syntax Checking of the test scripts

After the implementation of every test case, it is necessary to check the syntax and some potential bugs such as a variable or an object not having been defined. When developing the automated testing tool, pylint [19] was used for syntax error checking. However, pylint is only a simple syntax error checker, it is not possible to debug and locate bugs with the pylint tool. This is a significant drawback in this automation test tool. Additionally, when writing code the developers can utilize the automatic syntax features offered by Emacs. However, developers and testers sometimes have to dig deeply into the code to debug the scripts. Usually, these kinds of bugs will be found during integration of the test scripts into the regression test suite. Investigations are currently being performed concerning use of an integrated development environment (IDE) such as Eclipse (see chapter 2 of [29]).

4.2.7 Version Control of the test scripts

In Ericsson WRBS I&V, ClearCase (more details regarding ClearCase can be found in [21]) is used to manage different versions of scripts. To keep track of every

build of scripts, all the modifications will be made as a branch. Only after testing by the integration team and testing with several RBSs with different configurations will the modified version be merged into the main branch, then delivered to the users (normally, here “users” means the testers). The goal of this version control system is that it will be much easier for the script developers to maintain these test scripts and keep track on all modifications. This will be especially important when the test suite grows to hundreds of cases, as understanding each branch’s modifications will be very helpful when trouble shooting.

Chapter 5 Implementation and Integration

5.1 Automated Test Tool development with TMAP

In Chapter 3, the TMAP approach was introduced. The lifecycle model was introduced in Chapter 4. In this section, the four parts of the TMAP will be described as they actually relate to the actual automated test tool development effort of this thesis project. These four parts of TMAP are:

Lifecycle	In this thesis project, the goal is to reduce the software testing time - in order to reduce costs and improve the test efficiency.
Techniques	Object Oriented Programming and the open source programming language python have been used to implement the test scripts.
Infrastructure	The hardware infrastructure consists of: an RNC simulator, an MS simulator, an RBS, RDBTH, and FCCU. The details of these can be found in [14], [15] , [16], [17], and [18] respectively. The test target is the software running on an Ericsson WCDMA RBS model [16].
Organization	The automated testing tool development was carried out by a team leader, test developers, and integrators.

5.2 Automated Test Tool Implementation

5.2.1 Development Preparation

During the software development process, the planning & control phase carefully specifies the preconditions. Since well organized test cases simplify the implementation phase, a subset of all the automated test case candidates were selected and carefully described by the test case design team.

The test cases were designed and organized as a so called Function Verification Specification (FVS)[†]. When the test target is to be tested, then the testers follow the instruction in this FVS and check if the result of each test matches that stated in the FVS.

In the FVS, the whole test process will be specified step by step. The FVS states the precondition, test actions, and the test result for each test. In order to automate some of the test cases, the test developers usually have to read through the FVS, select the test cases that do not require any human intervention. This may require that the

[†] Note that the FVS is designed by the test analysis team.

test developer understand each test's steps; and if necessary run the test manually to make sure that there will not be any problem that could affect the test's result.

Based upon the FVS, the hardware and software necessary for the test environment should also be prepared. The software is a python module in a UNIX environment, the NITE platform software, and pylint syntax checking tool. All the necessary modules should be integrated in the tester's work station environment *before* the test development process starts. The hardware platform and software platform for the test infrastructure must match that assumed by the FVS; in fact, the FVS should explicitly say what infrastructure it assumes.

5.2.2 Test Script Development

5.2.2.1 Test Script Structure

A Test Script must include the following parts to form a complete test script:

- **Test Script Create and Revision History Record**

  At the very beginning of the script, it is important to record the developers' actions with regard to the creation and maintenance of the script. The record should include the developer's name, contact information, date of modification, and a short description of what changes have been made to the script. The purpose of this record is to keep track of each modification of scripts. Thus it is essential to record all the modification to the scripts. Additional details can be found in section 4.2.7.

- **Test Script Import Block**

  The test scripts build upon the NITE platform and are written in python. Python is an Object Oriented programming language, thus python supports objects. These objects are defined in terms of classes. Each class should be well designed and a common base class will be imported into each test script. Each test script should also import all the necessary utility classes and super classes.

- **setTcData Block**

  Each test case maps to a specific verification specification in the FVS. Each of these explicitly states the required test configuration. The setTcData() method of the NITE platform checks to see that the test's precondition is met. The precondition is defined in the FVS. All of the setTcData fields present in the example below are mandatory for all test scripts.

Example:

```
def setTcData (self) :
    return {

            'tcTag'              :'RBS_I&V_49E_D.N.0461',
            'tcName'             : 'DRL: TFR Selection, Incremental redundancy, 16 QAM',
            'comment'            : 'Tested',
            'status'             : 'RELEASE',
            'rbsType'            : 'ALL',
            'rbsSubType'         : 'ALL',
            'rbsConfig'          : 'HS',
            'rbsSw'              : 'P6-',
            'nbapVer'            : '6.9-',
            'niteVer'            : 'R34C-',
            'extConfig'          : 'RNCSIM,MSSIM,MDL'
            'transmissionType'       : 'ALL',
            'transmissionProtocol' : 'ALL'
            'rbsUeConnections'     : 'S1C1'
                    }
```

Note 1: Use only ‘status: RELEASE’ for test cases intended for regular use, i. e., regression testing.

Note 2: Only one test case shall be included in the tcTag field. As the result from each test must be reported per test case, this should provide a unique identifier for each test case. (The identifier will be generated by IBM Rational’s ClearCase tool [21]).

- **TcData field descriptions**

  The TcData fields describe the precondition(s) for the test case to be run, comments, and information about the test case information. These determine **if** the script *should* be run. Most of these fields support an “ALL” option, indicating that the script can handle any value in this particular field. Note that string values in python are delimited by single quotes.

  A field can contain multiple comma separated values. Specifying multiple values means that the precondition is satisfied if the actual value is any one of the values specified. For example, if the nbapVer field is “4.6, 6.9'”, this means the script applies to two cases; when the NodeB Application Protocol (NBAP) version is “4.6” or “6.9”.

  If a TcData field is empty, this is equivalent to the value “ALL”. However, to make the scripts consistent and easier to maintain, it is recommended that all fields are defined, thus this field should be explicitly set to “ALL”.

Information only fields (indicated as "**Informational**") should also be present in the tcData structure. If there is no specific information for this particularly Test Case, the field should be kept anyway, possibly with the empty string (") as value.

Table 1: Table of tcData fields

Field name	Description of field	Remarks
tcTag	The test case tag is a string indicating the FVS test case. Example: 'RBS_I&V_7N_R.1.N.1'	Informational
tcName	Test Case name, describing the test case's task Example: 'DRH: Setup RLS, SRB 3.4 kbps'	Informational
comment	The comment field describes deviations from the verification specification. TRs and CRs that have an impact on this test script must be listed in the comment field. This may also include information, such as modifications of the FVS.	Informational
status	Current status of this test case. Usual values: 'RELEASE' : The test script is finished and meant for regular use, i.e. regression testing. 'DEVELOPMENT' : The test scipt is currently under development and should **not** be used for regression testing.	Informational
rbsType	Type of RBS, out of all the WRBS types produced by Ericsson Supported values: 3018, 3101, 3104, 3106, 3107, 3116, 3202, 3203, 3206, 3206M, 3216, 3303, 3308, 3402, 3412, 3418, 3502, 3512, 3518, 3922, 3967, and ALL	
rbsSubType	RBS Subtype. Supported values: RBS2, RBS3, RBS4, and ALL	
rbsConfig	RBS configuration, i.e. number of sectors and carriers. Supported values: <sector>x<carrier> i.e: 1x1, 3x2, 3x3 etc.	"ALL" is only valid for the "sector x carrier" specification. That is, 'ALL' simply indicates all

	HS EUL_2MS_TTI TX_DIV MIMO RET ASC ALL‡ [7]	combinations of sectors and carriers, it does not concern the other values (such as HS). Thus, if the field's value is 'ALL', it only means all the sectorXcarrier attributes are included. The value "HS" indicates that the script needs an HSDPA capable RBS. EUL_2ms_TTI means enhanced uplink with 2 milisecond transmission intervals, thus a RBS must support this type of carrier. These features are defined in 3GPP specification.
rbsSw	RBS software version Example: 'P4-': Script supports revisions from project P4 onwards. '-P5': Script supports revisions up to and including P5. 'R2A-R2D' : Script supports revisions between R2A and R2D.	Supports both project revisions (such as P4, P5) and software revisions (such as R3A). For example, 'P5:R3A': Script will run if R3A revision of software, in the P5 project, is on the node. Moreover, a dash ('-') should be used if trying to specify that that the script supports up to a revision.
nbapVer	Nbap version Example 4.6-, 5.9-, 6.9- and ALL	
niteVer	NITE version supported by the script. Example: R<version number>, example R20. Means R20 of NITE is needed. -R<version number>, example -R15. Means NITE versions up to and including R15 are supported. R<version number>-, example R25-. Means NITE versions from R25 and later supported.	
extConfig	External tools that the test script needs.	If more than one value is

‡ HS stands for HSDPA resource required, EUL_2MS_TTI stands for enhanced uplink 2mili-seconds transmission interval, TX_DIV stands for Tx Diversity, MIMO stands for Multi antenna output/input, RET stands for Remote Electronic Tilt unit required, ASC stands for Antenna System Controller required, All means no special configuration required

	Supported values: RDBTH MDL MSSIM RNCSIM FCCU	given for this field, it means the test position configuration must contain equipment that satisfies all the values. That is, there is an AND relationship between the values. If MSSIM is included, then the script needs a MS Simulator. If RNCSIM is included, then the script needs and RNC simulator.
msConfig	MSIM Configuration, capabilities Supported values: HSDPA HSUPA, EUL MUE MBMS MIMO TX_DIV	
transmissionType	Transmission type, capability Supported values: Chan_STM1_E1 E1 E3 IP J1 STM1 T1 T3 ALL	
transmissionProtocol	The type of the transmission protocol(s) supported. Supported values: IP ATM IP_ATM ALL	IP_ATM is not supported until later in P6.
rbsUeConnections	Which cells needs to be connected to the UE. For example: a Test Case would require S1C1 and S1C2 to be connected. Supported values: S1C1	

	S2C1 S3C1 …	

- **Test Script Body**

  1st Step: Initialize the environment variables

  At the very beginning of every test script, initialization is required. This avoids possible conflicts between tests in a sequence of test cases in a test suite. At initialization, all of the variables will be initialized and set to an initial value. Since every test case begins with this step, the initialization method will usually will be placed in the test base class, so that every script can reuse this function.

  2nd Step: Initialize Pre-condition

  As specified in the FVS, some test cases need specific preconditions to be satisfied in order to run. These requirements are defined in the setTcData block. Additionally, a test case precondition may include some parameter changes, system configuration control, and so on. Thus, setting up the system to satisfy all of the preconditions will be the 2nd step in each test case script.

  3rd Step: Test Script body

  The test script's body contains all of the actions and specifies the expected results for this test case. Test developers should follow the FVS, preferably using methods provided by either the test base class or NITE platform. After each action, the script must check the result to confirm that the RBS system under test is working properly. A variety of check functions have been implemented in the test base class and NITE platform. A list of these check functions can be found in [30].

  Note that in different RBS software projects, i.e. P6 or P7, the RBS work procedure, interface, or parameters may have changed (from earlier versions). Therefore, test developers should be aware of such changes and carefully implement the test script with the necessary adaptation to these changes.

  Moreover, if the result check is unsuccessful, then an exception will be raised and the test case terminated after generating a log entry indicating the nature of the error for subsequent analysis.

- **Test Script End**

  Since the precondition and scenarios for each test case are different, when a test case execution has finished, the setup precondition, parameters and states must be cleaned up so that this test will not influence the next test case. As a test design principle, every action in the

test case should be stored in an action stack. At the end of each test case, the restore process will run the opposite action in reverse order according to the action stack. If the restoration procedure fails, an exception will be raised to warn the tester and a RBS restart operation should be performed.

Note that inverting the actions to return the RBS to a given state is preferable to performing a restart because in live network, the RBSs are not frequently restarted with those operations. The RBS's stable working status must be remained until something goes wrong.

- **Restart RBS**

  For the test cases that change the RBS configuration, these changes will not be saved after the test case's execution. In another words, any (new) settings are temporary. Ericsson's WRBS setting parameters are stored in a file called the "configuration version". Thus, changes in parameters will not be saved permanently in the configuration version file, but only modified for the current test case. After a RBS restart, the unsaved changes will be erased (i.e., they will not become permanent changes). This makes it possible to restore a RBS to a known initial state after a restart operation. The restart operation can be invoked by calling the NITE function AL_API_restartrbs(). After the RBS restart, to ensure that the next test case runs smoothly, this new test case first checks the RBS's status and will continue to execute the test case only after the RBS software has been re-loaded and all the relevant components in the RBS have been properly configured.

5.2.3 Legacy Regression Suite Usage

Within Ericsson's WRBS I&V department, the test scripts are integrated in a test suite called the "legacy regression suite". When testers upgrade an RBS with the latest software build, they start their testing with the legacy suite to verify that the legacy functions generate the same output as previously. Currently, the legacy regression suite takes serveral hours to run the entire suite (including time to log any errors).

Usually, the testers start the regression test in the afternoon and examine the results of this test suite the next morning. Given these test results, the testers will report their result(s) in a database and analyze all of the results. If there is any problem with the RBS software or with the test script itself, then a trouble report should be submitted indicating where there is a problem.

- **Legacy Suite Run-time Environment**

  The legacy suite will be executed in a UNIX (or Linux) environment. Moreover, the workstation will (pre-)load several load modules to ensure that the legacy suite can be executed correctly. These modules are:
  - Python 2.5.1
  - NITE Platform (latest version)

- NITE Scripts (latest version)
- Mobile station simulator with latest software version
- RNC simulator with latest software versions
- RBS Software, test version
- Test Environment, such as decoder, matched to RBS software

These modules should be carefully loaded and correctly chosen to ensure that the test process will be accurate. This may require that the human tester ensure that a specific version of the software is loaded into all of the external test equipment (including simulators).

- **Run Legacy Suite**

Depending on the node configuration and RBS type some test cases will be skipped as specified in the TcData description. Three examples of running the test suite are shown below:

Running the Whole Test Suite	python regr_TestSuite.py wrbs096 --testsuite=TestSuite \ --clearLogDir --nbapver=6.9
Run a Object of the Test Suite in this case mbdSuite	python regr_TestSuite.py wrbs096 --testsuite=mbdSuite \ --clearLogDir --nbapver=6.9
Run only one TC, in this case mbd_R4N3.Test	python regr_TestSuite.py wrbs096 –clearLogDir \ --nbapver=6.9 --testcase=mbd_R4N3.Test

NOTE: The --nbapver option defines the NodeB Application Protocol (NBAP) version and --clearLogDir will clear the logs which are stored in a temporary directory.

- **Failure Analysis**

1. The legacy suite is mainly used to do regression testing. The main goal is to ensure the quality of each software build. At a minimum the RBS has to work correctly, thus it is very important that testing starts by running the legacy test suite. Following this test suite, the testers check if any alarms occurred and that the data in the database is as expected. The testers also need to check the software feature license status (Ericsson WRBS software features are all based on license. The customers have to buy and activate a certain feature license if they want some functions to be run in theire network), before activating any of the features that they want to test. The operation is based on the software testing philosophy in Ericsson WRBS I&V. It is believed that all the features should be deactivated unless a single test case needs to test and then activate a certain feature.

2. Since the testing environment is a combination of different parts, such as Mobile Station simulator and RNC simulator, the testers may have to check the status of each of these test facilities. Usually, the testers try to ping the MS and RNC to see if the equipment (or emulation extension) are on-line.
3. After running a test suite, the failed test cases will be re-run for several times (which can be defined by the testers). As the testers will collect the test result after all the test process, the testers will analyze the results and re-run failed test cases to establish if these are reproducible errors.
4. The legacy test suite generates test results along with log file entries. During analysis, the testers utilize the log to identify if the apparent problem is in the RBS software or test script itself.

Chapter 6 Automation Tool Evaluation and Status

In this chapter, an evaluation of the automated tool development will be presented. Moreover, some future work will also be suggested. To evaluate the tools, the evaluation compares the cost of developing and maintaining an automated test environment versus the cost of doing the same testing manually or by relying on any faults being found (or not found at all) in other parts of the verification chain.

6.1 Test Coverage Improvement

In Ericsson's WRBS P7 Wiona Project, automated regression test was moved to the development (testing) phase. As a result a significant number of trouble reports (in the range of 50-70%) are detected internally within the RBS prior to delivery. This means that during the development phase, the automated legacy test suite can be very helpful finding the bugs concerning the legacy functionalities. Therefore, the number of test cases executed during development testing will significantly increase, as will the number of trouble reports generated. The legacy suite also has support for configurations other than those which were used (in the field) in order to increase the breadth of the test coverage, thus enabling factor testing to find *likely* faults. (For details of factor testing see section 9.5.2 of [20].)

6.2 Money saved by avoiding the error slipping into the next stage

There is a factor of two (based on earlier figures from the WBTS project[8]) in increased cost - if a potential problem propagates to the next stage. Therefore, if the problem remains undiscovered beyond the current stage, then the cost of finding this problem will increase even more. The automated test tool is designed to give a daily indication of the software's current status. During the development phase, legacy regression testing provides an indication of the software's status. Although, not all failures of legacy tests lead to trouble reports, the automated tests give the developers a "heads up" of the software's status, while providing project management with information related to the planned software deliveries to the subsequent stages of the development chain (or delivery to the end user).

Moreover, some of the faults that legacy testing may detect, might never have been found during manual verification (i.e., as manual verification or integration testing may not perform the same functional tests) or only found at a later stage in the development and testing chain (or after deployment at the customer's site, i.e., in the operator's network). Detecting these faults during the development phase by using automated tools greatly decreases the cost of errors. By stopping problems from reaching the live network, a large costs savings can be achieved and the risk associated with an upgrade is reduced.

6.3 Impact on Development

The RBS system is a large and complex system, thus it is very hard (almost impossible) to predict the impact on legacy code due to the introduction of new functionality due to either new software or hardware. Therefore development projects that introduce changes to legacy code will probably increase the number of trouble reports generated. This is especially true given that I heard someone say that in P7 "things are done to the hardware that it was not originally designed for". Whether this statement is true or not, the RBS system is a very large system and making changes that cause the hardware to operate in ways that it was not anticipated to operate are even more likely to cause unexpected problems. In this setting, an automated testing method gives the company the ability to test for unexpected impact(s) on the legacy functionality. If problems are detected early, it may be possible to correct the problem while the cost of correcting it is low.

A potential effect of integrating legacy testing with development testing is to increase the stability of the resulting system, as it more likely that both the new and legacy functions with both work – and that they will be compatible. Another potential effect is that developer can be more aggressive in making changes to the legacy code, rather than simply trying to make minimal patches – as they can quickly learn if a major reorganization of the code retains the legacy functionality while facilitating the addition of new functionality. If this effort is realized, then it may be possible to simplify the code while enhancing the functionality – thus resulting in fewer lines of code and fewer potential bugs!

6.4 Manual versus automated testing

Today the automated test tool automates more than 40% within all functional regression tests. The cost of the automated tool maintenance versus the manual regression testing of the automated test cases is an important metric. Manually executing the current test cases for every build requires more human resources than are available. However, with the automated tool these tests can be executed for a number of different configurations daily (i.e., several times per week) – and at a cost that is far less than what it would cost in human resources to conduct far fewer tests.

The number of trouble reports (concerning the RBS software and not the test scripts themselves) written after the introduction of the automated test environment has increased compared to the earlier manual case[4]. This is perhaps an indicator that the automated tool significantly increases the probability of finding potential bugs.

[4] Measurements show that the automated testing is up to twelve times more efficient than the manual regression test within the same time period.

6.5 Test coverage over time

Using the earlier manual regression test, all of the earlier functions could only be checked within the time of the entire project period. However, the automated test environment enabled regression testing to be executed for a large number of test cases in a single night, thus giving developers useful feedback the next day. This approached also reduced the time it takes for a new package build to reach a customer. Figure 8 shows the increasing number of test cases that are being used with this new automated test tool. (Note that the flatness in the curve for the July to August period reflects the summer vacation period for most employees; and the spurt of growth just before this period is due to lots of people trying to finish their work up before the start of their vacation.)

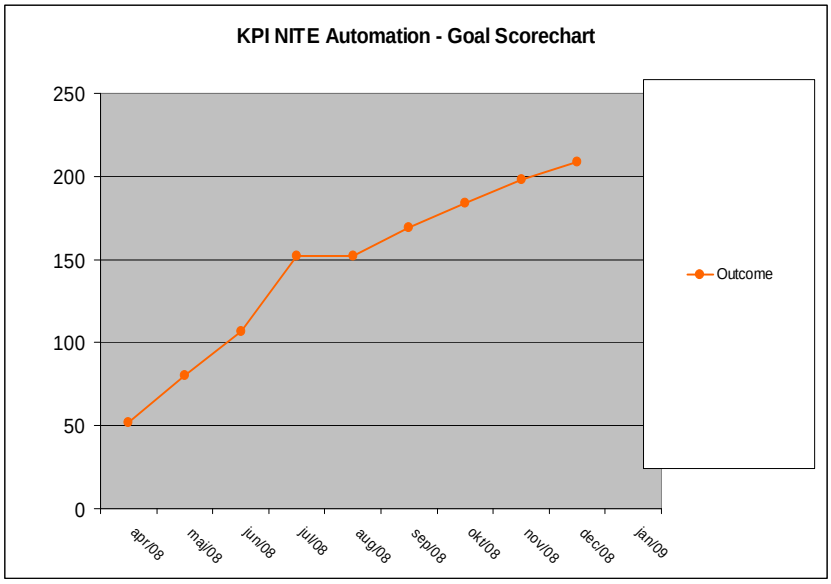

Figure 8: Increase in the number of test cases of the automated test tool as a function of time.

This thesis project accomplished its basic goal of automating part of the regression testing in Ericsson WRBS I&V. However, there remains additional work to be done – some of the most immediate work is described in the next section.

6.6 Future work

Today automated testing handles 40% of all test cases. This increases to 57% if all of the tests which require manual interaction are excluded. These manual test cases (17% of all test cases) need human intervention (such as the observation of an LED or pulling out/insertion of a plug-in module). In the near future, the development team is considering deploying a web camera to further increase the number of test cases which can be automated. It is anticipated that 70% of the existing manual test cases

can be automated. Thus the introduction of automated testing will supplants more than 70% of all the previous manual test cases, but has enabled these 70% of tests to be run daily - rather than only once over the entire testing cycle for a given release!

References

[1] Robert Oshana, DSP Software Development Techniques for Embedded and Real-time Systems, Newnes, ISBN 0750677597, 9780750677592, page 345, published, 2005

[2] Armstrong Process Group, Inc., Test Case Design with UML, Course description, Armstrong Process Group, Inc., New Richmond, Wisconsin, USA, 15 June 2006 http://www.aprocessgroup.com/training/pdf/APG_TestCaseDesign_CourseDesc_01_0804_v2_0.pdf latest visit 2009.02.01

[3] Ericsson, WRBS Regression Function Verification Specification, June 2008 (confidential document)

[4] Yuanfang Cai, Sunny Huynh, Tao Xie: A Framework and Tool Supports for Testing Modularity of Software Design http://www.cs.drexel.edu/~yfcai/Papers/ASE2007.pdf lastest visit 2009.02.02

[5] Martin Pol, Ruund Teunissen, and Erik van Veenendaal, *Software Testing – A Guide to the Tmap Approach*, ISBN 0-201-74571-2, published by Person Education Limited, UK, 2002

[6] Erik van Veenendaal and Martin Pol, "A Test Management approach for structured testing", Published in *Achieving Software Product Quality*, Erik van Veenendaal and Julie McMullan (eds.), UTN publishers, Den Bosch, The Netherlands, 1997. http://www.improveqs.nl/pdf/tmap.pdf

[7] Gunnar Kalsson, NITE TestScript Design Rules, 1/102 60-CXC 124 1020+ Uen, Ericsson Internal Document

[8] Summary of WBTS Project , Petter Isaksson, NITE Script Developer, 2005.11.10

[9] Pierr Jahan and Denis Degioanni, Third Generation Partnership Project (3GPP) web site, http://www.3gpp.org, 2008, last accessed: 2009.01.31.

[10] Harri Holma and Antti Toskala (Editors), *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications*, Wiley Technology Publishing, First edition, June 2000, 344 pages, ISBN-10: 0471720518 and ISBN-13: 978-0471720515.

[11] Erik Dahlman, Stefan Parkvall, Johan Skold, and Per Beming, *3G Evolution, Second Edition: HSPA and LTE for Mobile Broadband*, Academic Press, Second edition, October 2008, 648 pages, ISBN-10: 0123745381 and ISBN-13: 978-0123745385.

[12] Rosaline Makar, "Get started with unit and component testing using IBM Rational tools: A comprehensive guide for unit and component testing",

IBM, on-line tutorial, 11 October 2007, http://www.ibm.com/developerworks/edu/ws-dw-ws-testing.html

[13] Dimitri van Heesch, Doxygen: Source code documentation generator tool, Web page, Last modified 2 January 2009. http://www.stack.nl/~dimitri/doxygen/

[14] RNC simulator User Guide (TietEnator, confidential) Lastest release 2008.11

[15] Test Mobile Maual Book (confidential) Latest release 2009.2

[16] ERICSSON WRBS3000 User Guide (confidential) Latest release 2009.4.6

[17] RDBTH User Guide, 1553-CAL 115 0784, (confidential) Latest release 2009.3.17

[18] FCCU User Guide (Ericsson product, confidential) Latest release 2008.8

[19] Python community, "pylint 0.16.0: python code static checker", web page, Python Software Foundation, http://pypi.python.org/pypi/pylint - last accessed 2009.02.01

[20] Daniel Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson Education, 2004, 590 pages, ISBN 0-201-70945-7, 9780201709452.

[21] IBM Clear Case introduction, http://www-01.ibm.com/software/awdtools/clearcase/features/index.html?S_CMP=rnav, last vist 2009.03.05

[22] Standardization and related activities -- General vocabulary, ISO/IEC Guide 2:2004 (replaced ISO/IEC Guide 2:1996, that had replaced ISO/IEC Guide 2:1991), 8th Edition (Trilingual), International Standards Organization, Geneva, Switzerland, 2004, 60 pages.

[23] Yang Bin, Quality Assurance, Masters Thesis, Department of Communication Systems, Royal Institute of Technology, Stockholm, Sweden, work in progress.

[24] Christine B. Tayntor, *Six Sigma software development*, CRC Press, 2002, 322 pages, ISBN 0849311934, 9780849311932.

[25] Capers Jones, *Applied Software Measurement*, 3rd edition, McGraw-Hill Professional, 2008, 662 pages, ISBN 0071502440, 9780071502443.

[26] W. H. C. Bassetti, William E. Lewis, and Gunasekaran Veerapillai, *Software Testing and Continuous Quality Improvement*, 2nd Edition, CRC Press, 2004, 560 pages, ISBN 0849325242, 9780849325243.

[27] Stephen H. Kan, *Metrics and Models in Software Quality Engineering*, 3rd edition, Addison-Wesley, 2003, 528 pages, ISBN 0201729156, 9780201729153.

[28] Peter Farrell-Vinay, *Manage Software Testing*, CRC Press, 2008, 600 pages, ISBN 0849393833, 9780849393839.

[29] Jeff Younker, Foundations of Agile Python Development, Apress, 2008, 416 pages, ISBN 1590599810, 9781590599815

[30] NITE Scripts Function list (Ericsson Confidential) Latest release 2009.4.19

TRITA-ICT-EX-2009:4

www.kth.se