Automating a test method for a hybrid test environment

by

Tobias Eiderbrant

LIU-IDA/LITH-EX-A--10/024--SE

2010-06-09
Final Thesis

Automating a test method for a hybrid test environment

by

Tobias Eiderbrant

LIU-IDA/LITH-EX-A--10/024--SE

2010-06-09

Supervisor: Anders Rosell
Ericsson AB
R&D Linköping

Examiner: Petru Eles
Linköpings Universitet
Department for Computer and Information Science
Abstract

Ericsson has a very big and expensive test environment with a lot of GSM AXE equipment. In order to decrease the cost of testing Ericsson has developed a combination of simulated and real hardware, the Hybrid Test Environment (HTE). There is no formal supervision and testing of the HTE system today and this has left the HTE system unstable and the testers have been avoiding using HTE. It is important for Ericsson that the confidence for HTE will increase.

The goal of this thesis is to produce a method for testing the HTE system. An automated test tool has been implemented in order to monitor and test the HTE system. During the two weeks that the test tool has been operational it has discovered 4 servers in 3 different HTE rigs that malfunctioned. These servers were fixed and were operational before the end-users could discover any problem.
Acknowledgement

I would like to express my thanks to my colleagues at Ericsson in Linköping, especially my supervisor Anders Rosell, my manager Marie Sollén and all co-workers in the STE team, for making this thesis possible. I would also want to thank my examiner Prof. Petru Eles at IDA, Linköping University.

Last but definitely not least I want to thank my family for their love and support.
1 Introduction ............................................................................................................. 5
  1.1 Background ....................................................................................................... 5
  1.2 Purpose ........................................................................................................... 5
  1.3 Method ........................................................................................................... 5
  1.4 Requirements ................................................................................................. 6
  1.5 Limitations ..................................................................................................... 6
  1.6 Target audience .............................................................................................. 6
  1.7 Outline ........................................................................................................... 6

2 Theoretical background ........................................................................................... 9
  2.1 GSM System .................................................................................................. 9
    2.1.1 Overview ................................................................................................. 9
    2.1.2 Base Station Controller ......................................................................... 10
    2.1.3 Simulated Environment Architecture ............................................... 11
    2.1.4 Hybrid Test Environment ..................................................................... 12
    2.2 Computer system testing ............................................................................. 14
      2.2.1 Regression testing ............................................................................... 14
      2.2.2 Sanity testing ..................................................................................... 14
    2.3 Automated testing ....................................................................................... 14
      2.3.1 Keyword-driven testing ..................................................................... 16
    2.4 Ericsson Test Harness Core ....................................................................... 16
    2.5 Programming language .............................................................................. 16
      2.5.1 Python ................................................................................................. 16
      2.5.2 PHP .................................................................................................... 16

3 Evaluation ............................................................................................................ 17
  3.1 HTE ............................................................................................................... 17
    3.1.1 IT environment ...................................................................................... 17
    3.1.2 AXE hardware ....................................................................................... 18
    3.1.3 AXE Software ....................................................................................... 18
  3.2 Automated tests for HTE ............................................................................. 18
    3.3 Test tool architecture ............................................................................... 19
      3.3.1 Requirements ...................................................................................... 19
      3.3.2 Test bench .......................................................................................... 19

4 Implementation ................................................................................................... 23
  4.1 Programming language .............................................................................. 23
    4.1.1 Test bench ............................................................................................ 23
    4.1.2 Configuration and logging modules .................................................... 23
  4.2 Implementation of the test tool .................................................................. 24
    4.2.1 System overview ............................................................................... 24
    4.2.2 Configuration module ......................................................................... 24
    4.2.3 Logging module .................................................................................... 28
    4.2.4 Test bench daemon ............................................................................. 30
    4.2.5 Database ER diagram ......................................................................... 32
  4.3 Version control ............................................................................................. 33
  4.4 Backup .......................................................................................................... 34

5 Evaluation of test tool implementation ............................................................... 35
Figure 1, GSM system overview .............................................................................................................9
Figure 2 Block diagram of an BSC ..........................................................................................................11
Figure 3 Block diagram of HTE ...............................................................................................................13
Figure 4 Overview of the test system .....................................................................................................24
Figure 5 Illustration of test cases ..........................................................................................................25
Figure 6 Illustration of test case commands ..........................................................................................25
Figure 7 Illustration of the test classes ..................................................................................................26
Figure 8 Illustration of host configuration ............................................................................................26
Figure 9 Illustration of the schedule configuration ................................................................................27
Figure 10 Illustration of main configuration ..........................................................................................27
Figure 11 Illustration of the logging module with verdict PASS ............................................................28
Figure 12 Illustration of the extended log with verdict PASS .................................................................29
Figure 13 Illustration of failed test cases with the verdict UNRESOLVED ........................................29
Figure 14 Illustration of test cases with verdict FAIL ..........................................................................29
Figure 15 Illustration of extended log with test case with verdict FAIL ..............................................30
Figure 16 Illustration of test bench data flow .......................................................................................31
Figure 17 Illustration of test case ER diagram ......................................................................................32
Figure 18 Illustration of HTE host ER diagram ....................................................................................32
Figure 19 Illustration of schedule ER diagram ......................................................................................33
Figure 20 Illustration of configuration ER diagram .............................................................................33
1 Introduction
This thesis was performed at Ericsson AB as a part of a Masters degree in computer science and technology.

1.1 Background
Ericsson is a provider of telecommunications equipment and related services to mobile and fixed network operators globally. Over 1,000 networks in more than 175 countries utilize Ericsson network equipment and 40 percent of all mobile calls are made through Ericsson systems.

Ericsson has a very expensive test environment with a lot of GSM hardware. In order to provide a cost efficient solution a part of the hardware is simulated. The Hybrid Test Environment (HTE) is partly real GSM hardware and partly simulated in Scalable processor architecture (SPARC) and X86 server environment. A problem with HTE is that it is quite unstable and especially when the HTE system is left unused for a period of time.

It is strategically important for Ericsson that the stability for HTE will increase.

1.2 Purpose
There are no test activities regarding the HTE system in use today at Ericsson. Many of the problems in HTE are discovered by Ericsson’s internal customers and therefore the confidence for HTE is low. With low confidence the HTE customers will turn to real hardware instead and the total cost for testing will increase.

The aim of this thesis is to answer the following questions.

- Which test method is suitable for testing the Hybrid Test Environment?
- Can the test method be automated in the current IT environment?

1.3 Method
The work was divided into the following four phases.

1. Pre-study:
A lot of internal Ericsson documents were studied in order to understand the GSM and HTE systems. It is a complex system with a lot of test areas to cover and therefore a considerable amount of time was spent studying these areas.

Different test methods were studied in order to select an appropriate method.

2. Evaluation of test methods:
In the evaluation phase different test methods were evaluated and an appropriate
The selected test method was implemented and extensive tests were performed.

4. Evaluation of implementation:
In this phase an evaluation of the test method implementation was made.

1.4 Requirements
In discussions with the supervisor at Ericsson the following requirements were found to be relevant.

- **Requirement 1** - The resulting application shall be module based.
- **Requirement 2** - Support for regression test shall be included.
- **Requirement 3** - The source files shall be version controlled.

1.5 Limitations
The thesis will focus on testing of the Hybrid Test Environment which is a part of the test environment of GSM. The thesis will not cover testing of the other nodes in the GSM system.

1.6 Target audience
The target audience is anyone with basic knowledge of computer science.

1.7 Outline
1. Introduction
In this chapter the background, the purpose, the method and the requirements for this thesis are stated

2. Theoretical background
In this chapter the theoretical background of the GSM and HTE systems are described. The simulated environment architecture, system and automated testing and programming languages are also discussed.

3. Evaluation
In this chapter the Hybrid Test Environment, test method and the test tool architecture are evaluated.

4. Implementation

In this chapter the implementation of the test tool and its modules are discussed. The choice of programming language, version control and backup are described.

5. Evaluation of the test tool implementation

In this chapter the test tool implementation is evaluated.

6. Results

In this chapter the results from the implemented test tool are presented.

7. Conclusions and future work

In this chapter the conclusions of this thesis are presented. Possible future work is also discussed.
2 Theoretical background
This chapter includes theoretical background about the GSM system, the Hybrid Test Environment system and test methods.

2.1 GSM System
GSM is a digital cellular telephony system where Ericsson is the largest supplier. Even though the technology is over twenty years old Ericsson is still developing new hardware and software for the GSM system (Eberspächer & Vögel, 1999).

The Ericsson GSM system is based on the Automatic Cross-Connection Equipment (AXE) platform. AXE is a circuit switched digital telephone exchange initially intended for the Public Switched Telephone Network (PSTN) (Eberspächer & Vögel, 1999).

2.1.1 Overview
GSM consists of several nodes that have a hierarchical structure as seen in Figure 1.

Figure 1, GSM system overview
Mobile Services Switching Centre (MSC) performs control functions for circuit mode traffic such as authentication, location registers and equipment identity register (Ericsson, 2006).

Gateway MSC (GMSC) is the gateway towards the outside world, and controls calls to and from other telephony and data systems. The GMSC and the MSC are almost the same type of node but one major difference is that MSC is not connected to the outside world. The outside world is in this case other mobile telephony systems or the Public Switching Telephony Network (PSTN). The difference is only in the software of the MSC and GMSC (Ericsson, 2006).

Base Station Controller (BSC) manages all of the radio related functions of a GSM network. It is a high capacity switch that provides functions such as Mobile Station (MS) handover, radio channel assignment and collection of cell configuration data. Each MSC can control a number of BSC’s (Ericsson, 2006).

Base Transceiver Station (BTS) controls the radio interface towards the MS. It comprises the radio equipment such as transceivers and antennas which are needed to serve each cell in the network. Each BSC can control a number of BTS´s (Ericsson, 2006).

Serving GPRS Support Node (SGSN) forwards IP packets traffic to and from the MS that is attached within the SGSN service area. It provides services like packet routing, ciphering, authentication, session and mobility management (Ericsson AB, 2006).

Gateway SGSN (GGSN) is the gateway towards the outside world. The GGSN is responsible for routing IP packet traffic to the correct SGSN (Ericsson, 2006).

2.1.2 Base Station Controller

The BSC is, as mentioned in 2.1.1, a part of the GSM system and development is currently ongoing. The majority of the BSC testing is performed at Ericsson in Linköping. This chapter will present an overview of the architecture in a BSC (Tisal, 1997).

The BSC controls a major part of the radio network. Its most important task is to ensure the highest possible utilization of the radio resources. A BSC may be implemented on the AXE 10 platform or the AXE 810 platform (Ericsson, 2006).

The BSC consists of several subsystems, and the following three is described:

- The Central Processor (CP) is a major part of the control system in a BSC.
- The Regional Processor (RP) stores and executes the regional software. This software handles routine tasks, such as scanning hardware. The RP is controlled by the CP and
connected to the CP via the regional processor bus (RPB).

- The Input/Output (I/O) System is the human computer interface. The I/O system is connected to the CP via the Inter Platform Network (IPN).

The BSC hardware is depicted in Figure 2.

![Figure 2 Block diagram of an BSC](image)

### 2.1.3 Simulated Environment Architecture

Simulated Environment Architecture (SEA) provides the possibility to simulate AXE nodes in Linux based X86 or Solaris based Sparc servers.

There are three main reasons to simulate AXE hardware in SEA.

1. Develop software before the hardware is ready, thus gain valuable time to market.

2. Simulate the hardware in order to test the software is cheaper than executing the tests in real AXE hardware.
3. **SEA** has a powerful debugging functionality. It is possible to add breakpoints in the real software and single step the simulated environment in order to debug possible errors.

SEA is a powerful test solution but there exist limitations:

1. It is not a recommended solution when test cases require high load.

2. It is not recommended to use SEA when executing critical real time test cases.

### 2.1.4 Hybrid Test Environment

The Hybrid Test Environment system is a cost efficient partly simulated test solution.

#### 2.1.4.1 Architecture

HTE consists of one part real AXE hardware and one part simulated in SEA. The SEA session is executed in a X86 Linux or a Sparc Solaris equipped server as shown in Figure 3.

The RP´s and the cabinet where they reside are real AXE hardware while the rest of the BSC is simulated in SEA.
2.1.4.2 Advantages and limitations

Ericsson calculates that the cost of a HTE rig is 20% of the cost of a real BSC. In test cases where the focus is on testing true RP behavior HTE is a recommended solution. The real AXE hardware has a limited debugging functionality whereas SEA has a powerful debugging functionality as described in 2.1.3. HTE has a flexible configuration. It is fast and easy to switch between CP or I/O models without altering the hardware.

Just like SEA, HTE has limitations. HTE has the advantage over SEA due to real RP’s, but HTE still is not a recommended solution for high load test cases or test cases based on critical real time. This is due to the simulated CP and I/O system.

2.1.4.3 Problems

The HTE solution is a relative new solution. The development started about 4 years ago, and HTE still has some stability issues. There is no pro active test activity of HTE at Ericsson today and thus many of the problems are detected by Ericsson internal customer. The problems are discussed in chapter 3.1.


2.2 Computer system testing
This section describes different test methods.

In order to be able to guarantee some level of quality for a computer system some type of testing must be performed. In system testing it is only possible to prove that faults exist in the system. It is not possible to prove that a system is fault free. The purpose of testing is to find as many of the faults as possible (Kaner et al 1999). The tester’s role is not to try to verify that the application is running correctly. The role is to try to find as many faults as possible.

2.2.1 Regression testing
Regression test is performed in two different ways. Common to both is the idea of reusing old tests (Kaner et al 1999):

1. If the tester finds an error and the problem is taken care of then the test case, that exposed the error, will be executed again. Adding variations to the test series in order to make sure that the fault that caused the error is fixed is also considered regression test.

2. Imagine the same problem as in the previous case, but executing a standard series of tests to make sure that the change didn’t disturb anything else. This is also considered as regression test.

Both types of tests should be executed after fixing an error (Kaner et al 1999). Because regression test often is repeated it is especially suited for automated testing.

2.2.2 Sanity testing
A sanity test is a narrow regression test that focuses on one or a few areas of functionality. It is used to determine that a small section of the system is still working after minor changes.

Sanity testing is a cursory testing; it is performed whenever a cursory testing is sufficient to prove that the system is functioning according to specifications. This level of testing is a subset of regression testing (Vijay 2007).

2.3 Automated testing
Automated testing is a well known technique (Fewster & Graham, 1999). There are several reasons to automate tests:

- Speed up testing.
- Allow tests to run more frequent.
• Reduce cost of testing by using less manual labor.
• Improve test coverage.
• Ensure consistency.
• Improve the reliability of testing.
• Allow testing by persons with less programming skills.

When designing an automated test tool there are many areas to consider in order to make the tool do the correct things for the people that use it. There are some guidelines to follow in order to avoid errors in the process of designing and implementing the test tool (Pettichord, 2001).

• **Reviewability:** It is important that the testers know what the system is testing and what the outcome is saying. There must be printouts and logs in order for the tester to analyze the test result in the correct way. All commands that were executed should be logged in order to analyze what the test system actually tested (Pettichord, 2001).

• **Maintainability:** It is not unlikely that the output of a system changes between different releases. This might force changes to the test tool. Therefore it is important to build an abstract layer in order to minimize the changes to the test tool in case of changes in the systems output. This can be implemented as libraries that are common in many different tests (Pettichord, 2001).

• **Integrity:** The testers must be able to trust the test system. Is it due to the test system, the system under test or the test case if the test case fails? It is not sufficient with the verdicts “PASS” and “FAIL”. There must be a third verdict included that communicates to the tester that the test tool failed and that it might not be the system under test that has failed. This verdict can be called “UNRESOLVED” (Pettichord, 2001).

• **Independence:** A test may be depended on a previous test and a human may be able to understand that something is wrong. It is much harder for a computer to “see” that this is the case. If a test fails and the succeeding test fails it is harder to determine where the problem is. Therefore it is important that a test case can be executed independently (Pettichord, 2001).

• **Repeatability:** It is important that the test is performed and executed in the same way every time that it is run. A test result is more valuable if it is repeated several times. If a test case fails then it is important that the error can be found the next time the test case is
executed (Pettichord, 2001).

### 2.3.1 Keyword-driven testing

Script based testing is based on test scripts that executes test cases. The script consists of a short program written in a program language that tests a predefined part of a system.

Keyword-driven testing is a testing technique that separates much of the programmer’s work of test automation from the actual test design. The test cases are implemented with keywords. The keywords are predefined words which are translated into automated test scripts (Buwalda, 2007).

### 2.4 Ericsson Test Harness Core

At Ericsson today, there exist a test automation framework, the Test Harness Core (THC).

THC is a distributed test automation system based on Common Object Request Broker Architecture (CORBA) technology, and hence aims to be independent of platform and application programming language. The communication between components in a distributed CORBA based system is carried out through interfaces specified in Interface Definition Language (Ericsson, 2010).

THC has a support organization at Ericsson but is developed by Cybercom in Östersund.

### 2.5 Programming language

In this section the programming languages that were selected for implementation are presented.

#### 2.5.1 Python

Python is an interpreted, interactive, object-oriented programming language which is implemented in the programming language C. Python is a high-level general-purpose programming language that can be applied to many different classes of problems. It has support for UNIX/Linux as well as Windows and the code written in Python is thus portable. It is very well documented on a variety of web pages and in books. Python is distributed as open source and will therefore be developed for future architectures and platforms (Python Software Foundation, 2010).

#### 2.5.2 PHP

PHP is a widely used general purpose scripting language that is especially suited for Web development. It has, just like Python, support for UNIX/Linux as well as Windows and the code written in PHP is thus portable. PHP is distributed as open source and will therefore be developed for future architectures and platforms (PHP Group 2010).
3 Evaluation
In this chapter HTE and its problems are discussed and categorized. The requirements for the test tool are stated, and finally a solution is chosen.

3.1 HTE
As mentioned in chapter 2.1.4.3 HTE has some stability issues. In order to identify as many issues as possible the problems must be categorized. The following problem classes were identified.

3.1.1 IT environment
Every HTE rig consists of at least one server, but the most common setup consist of two servers and at least one Ericsson AXE RP magazine. All HTE servers must have a predefined setup. The critical part of the configuration is:

- **Daemons** running on the servers such as:
  - Traffic Simulator System (TSS) daemon – This daemon is used to simulate traffic, air interface, mobile stations, and other AXE nodes.
  - Link handler daemon – This daemon is used to connect the serial RPB, simulated in SEA, to the real AXE RP.
  - Ethernet broadcast daemon – This daemon is used to connect to the Ethernet RPB, simulated in SEA to the real AXE RP.
  - RPBE Gateway daemon – This daemon is used for the IP traffic between the simulated CP and the real RP’s. The daemon is started when a RPBE HTE network is started i.e. it is only running when a HTE rig is activated and a SEA session is running.

- **IP configuration** – There are about 70 virtual network interfaces defined on the servers. The virtual interfaces are required by the TSS daemon and also by SEA in order to connect to the real AXE hardware. This setup is defined manually on top of the standard installation on the server.

- **Mapped network disks** – All software that is executed on top of the operating system is stored and accessed on separate file servers. The concept with centralized software is manageable but it also introduces risks. If a mapped drive fails on a server then the server will fail to complete its tasks.
Ericsson has a very large and complex IT environment. The environment is often updated and maintained and during these maintenance stops it is possible that interference affects the HTE environments. Due to this it is absolutely crucial to monitor and test-run the HTE part of the environment frequently e.g. nightly.

### 3.1.2 AXE hardware
There are several issues regarding the axe hardware that need to be monitored. These are summarized in the following list.

- The RP´s can fail and end up in a state where a time consuming manual hard reset is required. The tester must report this to the Ericsson test environment (BETE) help desk and personnel from the help desk must remove the RP board and plug it in again. This is a problem only related to HTE and not to the real BSC.
- In certain kind of setups a real MSC is used and therefore the link towards the MSC must be monitored.
- The magazine that contains the RP´s must be operational and working.

### 3.1.3 AXE Software
In order to configure the AXE node a configuration dump is built and loaded into the node. The build process of the dump is very complex and faults may be introduced. There are parts of the software that are critical for HTE. These are summarized in the following list.

- The RP software must have the correct version and type.
- The RP must be able to be activated.
- The Switching Network Terminal (SNT) must become active when the corresponding RP is activated.

### 3.2 Automated tests for HTE
One of the questions that this thesis shall answer is if it is possible to automate a test method for HTE.

The first decision was to decide if a test approach with test script was preferable. The advantage is that it is rather simple for a programmer to construct test scripts that executes the test cases. Test scripting has the advantage that the program flow can imitate the real execution in a simple manner. The disadvantage is that the maintenance cost may be rather high and may require a skilled programmer. One of the goals of this thesis was to make the resulting application as user
friendly as possible and that a user of less programming skill shall be able to administer the tests. This is possible by using a keyword-driven test method. The keyword-driven test method is suitable for automating test executions. It is quite easy to build a table of input commands to be executed in the HTE environment, retrieve the answer printout, and compare it against a predefined answer.

The users that are intended to use the test tool for HTE cannot be considered to be strong programmers. It is important that they intuitively can build a test case, or a test suite out of test cases, in order to test the HTE system. With keyword-driven tests they can do this.

### 3.3 Test tool architecture

In this section the architecture of the test tool will be discussed. Two test bench architectures will be discussed and one will be selected for implementation.

#### 3.3.1 Requirements

In addition to the requirements stated in chapter 1.4 some extra requirements had to be added in order for the test tool to fulfill the test objectives.

- **Requirement 4** - The test tool must have support for different test cases.
- **Requirement 5** - The test tool must have support for different suites of test cases.
- **Requirement 6** - The test tool must have support for automated execution of the test cases and test suites.
- **Requirement 7** - The test tool must have support for logging.
- **Requirement 8** - The test tool must have support for presenting the logs.
- **Requirement 9** - The test tool must be implemented in free software.

#### 3.3.2 Test bench

One important part of the test tool is the test bench. This is the sub system that performs the actual execution of the test cases. There are two different approaches to building the test bench. The choices are building a new test bench or using an existing test framework as a foundation of the test tool.

#### 3.3.2.1 Test Harness Core

There exists, as described in section 2.4, a framework for test automation at Ericsson. It is a large and complex environment with several subsystems.
There are advantages with THC, such as:

- Support for big and complex setups with many different telecom nodes.
- It has support for many different telecom nodes.

There are disadvantages with THC, such as:

- It may take a long time to get new features implemented if they are missing.
- It is sensitive for IT environment changes, see 3.3.2.3.
- Its complexity makes the execution slow.

### 3.3.2.2 Develop new application

There are advantages with developing new software:

- Develop the software to fit the system exactly.
- Develop the software to be small and fast.
- Implement new features when needed.

There are also disadvantages with developing new software:

- The development process takes time and costs money.
- The responsibility for maintenance and upgrades lies on the developer.

### 3.3.2.3 Selection of test bench architecture

In the test tool that is developed the test bench is rather small and easy to maintain and upgrade. Although THC is powerful and flexible it is also rather slow and sensitive for changes in the IT environment. In order to be able to log in to a server via SSH, THC had to know exactly how the prompt in the execution console should look like. The prompt differed between different operating systems and also between different releases of the same operating system.

In HTE there exist two different operating systems on approximately 15 different servers. The number of servers is currently increasing and if the objective of this thesis is met the number of servers will increase even more. It is not a good solution to alter the expected prompt for every server that will be logged in to. It is more fail safe to implement a general solution which is not sensitive for IT related changes.

THC is a relative slow system. In order log in to a server and execute the UNIX command “ls –l /proj/steteam”, it took 50 seconds for the system to start executing and another 50 seconds for the THC system to execute the command.
The decision was made to develop new software for the test bench. These were the main reasons:

- The sensitivity for the surrounding IT environment. Sensitivity regarding the IT environment is the most important reason for the instability of HTE. Such sensitivity should not be introduced in the test tool system.

- The test bench is not large and complex.

- The time spent for development and maintenance will be moderate.
4 Implementation
The implementation of the test tool is described in this chapter. Implementation of the different modules, the test bench and the configuration and logging modules are described.

4.1 Programming language
In order to fulfill requirement 9, stated in chapter 3.3.1, a programming languages have to be chosen. The programming language has to be free and well suited for the application.

The test tool was divided into two main parts, the test bench and the configuration and logging modules

4.1.1 Test bench
The test bench handles all communication to and test of the HTE rigs.
In the test bench there are several requirements for the programming language:

- It shall be free software.
- It has to support threads.
- It has to be able to interact with MySQL.
- It must be an object oriented language.
- It must be stable and widely used.

Python fulfills all of the requirements above and is therefore selected for the implementation of the test bench.

4.1.2 Configuration and logging modules
In the configuration module all configuration is stored in a data base and it is possible to alter the configuration via a web page.
In the configuration and logging modules there were several requirements for the programming language:

- It shall be free software.
- It has to be able to interact with MySQL.
- It must be an object oriented language.
- It must be suitable for web development.
- It must be stable and widely used.
PHP fulfills all of the requirements above and is therefore selected for the implementation of the configuration and logging modules.

4.2 Implementation of the test tool
This section describes the test tool implementation.

4.2.1 System overview
The test tool consists of three main sub systems.

- **The test bench** - Performs the actual testing of the HTE system.
- **The configuration module** – Handles all configurations that control the test bench.
- **The logging module** – Handles all resulting test and debug logs.

Figure 4 illustrates the test system.

![Figure 4 Overview of the test system](image)

4.2.2 Configuration module
In the configuration module it is possible to control the test bench daemon. The configuration is divided into four parts, test case configuration, host configuration, schedule configuration and main configuration. The four different configuration modules are implemented in PHP and are executed on an Apache web server which is connected to a MySQL database server.

4.2.2.1 Test case configuration
The test cases configuration consists of a connection type, host type information and a test case command. The connection type can be SSH or telnet depending on the test case command. The
host type information can be SEA, TSS or both. The test case command is a command that will be executed on the server, in SEA or in TSS. A priority is set for each test case in order to verify the execution order. The test bench application will execute the test cases in an ascending order starting with a test case with low priority. It is possible to make the test cases execute in parallel mode. This option is set in the parallel option. If the test case returns with the verdict fail then a notification mail is sent to the mail address specified in the notify option. This is illustrated in Figure 5 and Figure 6.

![Figure 5 Illustration of test cases](image)

![Figure 6 Illustration of test case commands](image)

There may be many specific test cases to cover all testing for a specific area. For instance there are nine test cases included in the test class “IT Environment RPBE”. It is easier to administrate one test class instead of several test cases. Imagine the scenario where five HTE rigs shall be scheduled to run. All nine test cases in “IT Environment RPBE” shall be included and thus $9 \times 5 =$
45 test cases need to be added. If the test classes can be added instead then just 5 HTE rigs need to be added with 1 test class each.

<table>
<thead>
<tr>
<th>ID</th>
<th>Test Case Class</th>
<th>Test Case</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>IT Environment RPBE</td>
<td>HTE_check_routing_table_tss</td>
<td>☐</td>
</tr>
<tr>
<td>11</td>
<td>IT Environment RPBE</td>
<td>HTE_check_routing_table_sea</td>
<td>☐</td>
</tr>
<tr>
<td>10</td>
<td>IT Environment RPBS</td>
<td>check_tss_daemon</td>
<td>☐</td>
</tr>
<tr>
<td>9</td>
<td>IT Environment RPBS</td>
<td>check_ose_linkhandler_rpbs</td>
<td>☐</td>
</tr>
<tr>
<td>8</td>
<td>IT Environment RPBE</td>
<td>check_ethbrdcs daemon_linux</td>
<td>☐</td>
</tr>
<tr>
<td>7</td>
<td>IT Environment RPBE</td>
<td>HTE_check_ipvconfig_sea</td>
<td>☐</td>
</tr>
<tr>
<td>6</td>
<td>IT Environment RPBE</td>
<td>check_tss_daemon</td>
<td>☐</td>
</tr>
<tr>
<td>5</td>
<td>IT Environment RPBS</td>
<td>check_net drive</td>
<td>☐</td>
</tr>
<tr>
<td>4</td>
<td>IT Environment RPBS</td>
<td>check_ssh_connection</td>
<td>☐</td>
</tr>
<tr>
<td>3</td>
<td>IT Environment RPBE</td>
<td>check_ipvconfig_tss</td>
<td>☐</td>
</tr>
<tr>
<td>2</td>
<td>IT Environment RPBE</td>
<td>check_net drive</td>
<td>☐</td>
</tr>
<tr>
<td>1</td>
<td>IT Environment RPBE</td>
<td>check_ssh_connection</td>
<td>☐</td>
</tr>
</tbody>
</table>

Figure 7 Illustration of the test classes

4.2.2.2 Host configuration

The host configuration consists of a hostname which is the HTE or STE name and the corresponding hostnames for the SEA server and TSS server. The active option is set if a HTE host shall be available for scheduling. The active option only affects new schedules. If a HTE host is scheduled and the active option is deselected the test case will be executed but it is not possible to schedule a new test case. There are two RP Bus types and this must be added in the RP BUS option.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>SEA host</th>
<th>TSS host</th>
<th>RP BUS</th>
<th>Active</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>HTE_LOGIN_FAIL</td>
<td>selibx101</td>
<td></td>
<td>RPBE</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>34</td>
<td>HTE_FAIL_TEST</td>
<td>SEATEST</td>
<td>TSSTEST</td>
<td>RPBE</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>32</td>
<td>HTE753 M@H</td>
<td>selibx158</td>
<td>selibx158</td>
<td>RPBS</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>27</td>
<td>HTE604</td>
<td>selibx159</td>
<td>selibx159</td>
<td>RPBS</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>26</td>
<td>HTE601</td>
<td>selibx160</td>
<td>selibx160</td>
<td>RPBS</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>25</td>
<td>HTE760</td>
<td>selibx122</td>
<td></td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>24</td>
<td>HTE759</td>
<td>selibx113</td>
<td>selibx341</td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>23</td>
<td>HTE758</td>
<td>selibx157</td>
<td>selibx342</td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>22</td>
<td>HTE756</td>
<td>selibx112</td>
<td>selibx345</td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>21</td>
<td>HTE755</td>
<td>selibx156</td>
<td>selibx344</td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>3</td>
<td>HTE757</td>
<td>selibx161</td>
<td>selibx343</td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
<tr>
<td>1</td>
<td>HTE199</td>
<td>selibx089</td>
<td>selibx090</td>
<td>RPBE</td>
<td>1</td>
<td>☐</td>
</tr>
</tbody>
</table>

Figure 8 Illustration of host configuration

4.2.2.3 Schedule configuration
The schedule configuration consists of date, time, date type, day in week and if the test case shall run as soon as possible. Date, time and date type is always added to the configuration. Date type can be “Now”, “Once”, Weekly” or “Daily”. Day in week is only added if “weekly” is selected as date type. “Run ASAP” is selected if “Now” is selected as date type. “Run ASAP” is used as an internal state machine in the test bench.

<table>
<thead>
<tr>
<th>ID</th>
<th>Date</th>
<th>Time</th>
<th>Date type</th>
<th>Day in week</th>
<th>Run ASAP</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>2010-05-09</td>
<td>00:30:00</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9 Illustration of the schedule configuration

This however is concealed from the user due to the complexity in the different choices. The scheduling is performed in the main configuration.

4.2.2.4 Main configuration

The configurations in the sections above are added to the main configuration.

<table>
<thead>
<tr>
<th>ID</th>
<th>STE/HTE Name</th>
<th>Test Case name</th>
<th>Schedule</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>427</td>
<td>hte604</td>
<td>Check tss daemon</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>426</td>
<td>hte604</td>
<td>Check OSE Linkhandler RPBS</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>425</td>
<td>hte604</td>
<td>Check /proj/stereum network drive</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>424</td>
<td>hte604</td>
<td>Check ssh connection</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>423</td>
<td>hte601</td>
<td>Check tss daemon</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>422</td>
<td>hte601</td>
<td>Check OSE Linkhandler RPBS</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>421</td>
<td>hte601</td>
<td>Check /proj/stereum network drive</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>420</td>
<td>hte601</td>
<td>Check ssh connection</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>419</td>
<td>hte760</td>
<td>Check routing table RPBS TSS</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>418</td>
<td>hte760</td>
<td>Check routing table RPBS SEA</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>348</td>
<td>hte199</td>
<td>Check /proj/stereum network drive</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
<tr>
<td>347</td>
<td>hte199</td>
<td>Check ssh connection</td>
<td>Daily - 00:30</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10 Illustration of main configuration

As seen in Figure 10 the STE/HTE name, test case or test class and existing or new schedule is added to the main configuration. All of the test cases in the test class will be added to the configuration if a test class is selected and added. In the figure above “hte604”, for instance, is added to the configuration. The test class “IT environment RPBS” was selected and four test cases were added to the configuration.
In the schedule section there is a possibility to reuse an existing scheduled date or create a new. As seen in the figure above all of the scheduled events are using the same scheduled date. If a new date is selected then just add a date, time and date type in the recurrent drop down menu. If “weekly” is selected as date type then a weekday must be added. The recurrent date types “weekly” and “daily” are saved as existing schedule dates. The types “once” and “now” are only valid once and thus not saved.

4.2.3 Logging module

The logging module consists of a web page where the logs from a database are present in a user friendly manner.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Create Date</th>
<th>Verdict</th>
<th>View Extended Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>1349</td>
<td>Check ssh connection - HOST: sellbx343@hte757</td>
<td>2010-05-13 09:12:19</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1348</td>
<td>Check ssh connection - HOST: sellbx110@hte757</td>
<td>2010-05-13 09:12:18</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1347</td>
<td>Check /proj/stteam network drive - HOST: sellbx343@hte757</td>
<td>2010-05-13 09:12:18</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1346</td>
<td>Check /proj/stteam network drive - HOST: sellbx110@hte757</td>
<td>2010-05-13 09:12:17</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1345</td>
<td>Check IP RPBE TTS - HOST: sellbx343@hte757</td>
<td>2010-05-13 09:12:17</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1344</td>
<td>Check tss daemon - HOST: sellbx343@hte757</td>
<td>2010-05-13 09:12:16</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1343</td>
<td>Check IP RPBE SEA - HOST: sellbx110@hte757</td>
<td>2010-05-13 09:12:16</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1342</td>
<td>Check Ethernet Broadcast daemon - HOST: sellbx110@hte757</td>
<td>2010-05-13 09:12:16</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1341</td>
<td>Check routing table RPBE SEA - HOST: sellbx110@hte757</td>
<td>2010-05-13 09:12:15</td>
<td>PASS</td>
<td>View Log</td>
</tr>
<tr>
<td>1340</td>
<td>Check routing table RPBE TTS - HOST: sellbx343@hte757</td>
<td>2010-05-13 09:12:15</td>
<td>PASS</td>
<td>View Log</td>
</tr>
</tbody>
</table>

Figure 11 Illustration of the logging module with verdict PASS

All executions of test cases are logged and saved in the database; a small example can be seen in Figure 11. A description of the test case, a timestamp when the execution of the test case was started, a verdict and a “View Log” button are added to the table. An extended log can be viewed by pressing the “View Log” button. An example of the extended log can be seen in the figure below.
In this example the “Check tss daemon” test case has been executed on host seilbx343. The host is a part of the HTE rig hte757. Several steps in the execution chain can be viewed in this table. One important part is the log with the exact command that was, in this case, executed on the server. In the extended log, with the highest id number, there must be a log entry with a verdict, “PASS” or “FAIL”, which means that the test case has executed fully. If the verdict is missing the verdict in Figure 11 will be marked as “UNRESOLVED”. This means that the test case has not been fully executed. An example of test cases that have failed can be seen in Figure 13.

A test case can return the verdict “FAIL” as seen in Figure 14.

In the extended logs it is possible to track down the reason why the test cases returned the verdict FAIL. The logs contain a record of the person or persons that have received a notification mail. This can be seen in Figure 15.
4.2.4 Test bench daemon

The test bench daemon has several tasks:

- Run test cases which are scheduled to be run.
  The daemon is executed continuously and queries the database once every minute to check whether there is a task scheduled to run.

- If there is a scheduled test case the daemon performs the following tasks:
  - Get SEA and TSS host.
  - Check if the hosts are valid for execution.

- Connect to the target server via telnet or SSH where the test will be executed.
  - The daemon queries the database and checks if the connection shall be ssh or telnet.
  - Connect to the server or servers.

- Execute the test case on the target server.
  - Execute and collect the answer.
  - Retrieve the predefined expected value from the database.
  - Compare the result against a predefined value in the database.

- Log the result and debug data to the database.

The data flow can be seen in Figure 16.
Figure 16 Illustration of test bench data flow
4.2.5 Database ER diagram

In this section ER diagrams of the database structure are presented.

Figure 17 Illustration of test case ER diagram

Figure 17 illustrates the test case relations in the database.

Figure 18 Illustration of HTE host ER diagram

Figure 18 illustrates the HTE host relations in the database.
Figure 19 Illustration of schedule ER diagram

Figure 19 illustrates the schedule relations in the database.

Figure 20 Illustration of configuration ER diagram

Figure 20 illustrates the configuration relations in the database.

### 4.3 Version control

Requirement 3 is stating that all source files shall be version controlled. It is very important to keep track of all changes during the lifecycle of software. At Ericsson the version control system Rational ClearCase from IBM is used.

All source files were added to ClearCase and can be accessed from the ClearCase repository.
4.4 Backup

It is important to keep in mind that computer systems may fail. The database, where all configuration and logs are stored, will be replicated onto a backup server within certain time intervals.

The version control system is also replicated to backups.
5 Evaluation of test tool implementation

In this chapter the evaluation of the test tool implementation is presented. The requirements are validated and metrics are identified and presented.

5.1 Requirements on the test tool

In order to implement a test tool that fulfills the test objectives all requirements stated in sections 1.4 and 3.3.1 must be fulfilled.

**Requirement 1** – The resulting application shall be module based.

The resulting application is module based and consists of three major modules which are implemented as sub-modules.

**Requirement 2** – Support for regression test must be included.

There is support for regression test included. In chapter 2.2.1 regression test is described. It is possible to repeat the test several times and run a standardized test suite to make sure that the error is corrected.

**Requirement 3** – The source file shall be version controlled.

The source files are version controlled in the ClearCase version control system.

**Requirement 4** - The test tool must have support for different test cases.

The test tool configuration has support for different test cases.

**Requirement 5** - The test tool must have support for different suites of test cases.

The test tool has support for different suites of test cases. In the test tool configuration the test suits is also called test class.

**Requirement 6** - The test tool must have support for automated execution of the test cases and test suites.

It is possible to schedule the test case or test class to be executed on a specific date and time. It is also possible to make the test case or test class execute recurrently.

**Requirement 7** – The test tool must have support for logging.

The results are logged to a database and presented to the user.
Requirement 8 - The test tool must have support for presenting the logs. The results are logged to a database and presented to the user.

Requirement 9 - The test tool must be implemented in free software. The test tool is implemented in Python and PHP which are free software.

5.2 Metrics for the test tool
It is not trivial to exactly measure how good a test tool really is. One important aspect is the knowledge of what the objectives are and to measure the attributes related to the objects. When choosing the attributes it also important to select a limited amount, i.e. choose three or four that will give the most information about if the objectives are met (Fewster & Graham, 1999). The following test metrics have been selected in order to evaluate the test tool.

5.2.1 Maintainability
It is important to realize that software and computer systems change over time and develop the test tool according to it. The test tool should not be to difficult to change according to the changes in the surrounding environment (Fewster & Graham, 1999). This can be measured by looking at:

- Average elapsed time to update the test:

  The average time to update a test in the test tool is not very large. All test cases are based on executing a command and inspect at the result. It is easy to change this by adding a new test case command and execute the test again.

- How often is there a change in the system?

  In the IT environment this is related to answer printouts from the operating system on the server. It is not often an answer printout is changed.
  In the AXE environment it is not often the answer printout changes. It is more likely that a new command is added rather than a command is altered.
  When this happens the time for maintaining the test tool is low and thus the impact on the test tool maintenance time is low.

5.2.2 Reliability
The reliability of an automated test is related to its ability to give accurate and repeatable results (Fewster & Graham, 1999). This can be measured by looking at:
• The percentage of tests that fail because of the test themselves. This is where the test has not been able to execute fully and the verdict is recorded as “UNRESOLVED”

• The number of false negatives. This is where the test is recorded as “FAIL” but the actual outcome is correct. This could be due to incorrect expected results.

• The number of false positives. This is where the test is recorded as “PASS” but the actual outcome is erroneous. This could be due to incorrect expected results or incorrect comparison.

All three of the measurements are related to the expected outcome and the comparison in the test bench daemon. The actual answer of a command is rarely changed and it is easy to update the configuration in the test tool accordingly.

5.2.3 Usability

There may be different usability requirements on a test tool for different types of users. The time required for a skilled user will differ from a user with less skill. Usability must be measured in terms of the intended users (Fewster & Graham, 1999).

All of the users that will operate the test tool should have basic knowledge regarding Linux/UNIX and a good knowledge regarding the GSM AXE system.

The usability can be measured by:

• The time required to add a new test to the test tool.

• The training time needed for a user to become confident and productive.

The test tool was implemented with the intention to be easy to operate. It will require a few minutes to add a new test to the test tool. The training time needed to operate the test tool is very low.
6 Results
This chapter presents the results from the evaluation of the test tool in chapter 5.

6.1 Requirements on the test tool
In 5.1, Requirements on the test tool it was concluded that all of the requirements stated in this report were fulfilled.

6.2 Metrics for the test tool
Three different metrics were selected in chapter 5.2 and in this chapter the result will be presented.

6.2.1 Maintainability
The following metrics were identified:

- Average elapsed time to update the test:
  It is most likely that it is the test case command which is going to be updated. The time required to update the test case is 30 seconds.

- How often is there a change in the system?
  There are two different systems that can be altered. The IT environment and the AXE environment.

  **IT environment** – All of the tests in the environment consist of standard Linux and UNIX commands. The changes occur with years apart. It is more likely that the configurations will change in the HTE setups, but this is also done rarely and with some sense of control and early warning.

  **AXE environment** – All changes are done in different projects and are always well documented. The users will know in advance when a change will occur. This will occur a few times every year.

The conclusion is that changes occur rarely in both environments and the average time to correct the faults that may occur in the test tool will be minutes every year.
6.2.2 Reliability
The test tool has been executing test cases regarding the IT environment every night for two weeks. The nightly runs included 77 test cases on 15 servers. The servers are part of 9 different HTE rigs.

The following metrics were identified:

- The percentage of tests that fail because of the test themselves.
  The test logs were inspected and there were no test cases that indicated “UNRESOLVED”.

- The number of false negatives:
  The test logs were inspected and there were no test cases that indicated “FAIL” when the actual outcome should be “PASS”

- The number of false positives:
  The test logs were inspected and there were no test cases that indicated “PASS” when the actual outcome should be “FAIL”

6.2.3 Usability
The following metrics were identified:

- The time required to add a new test to the test tool.
  To add a new test case to the test tool will require 2 minutes.

- The training time needed for a user to become confident and productive.
  The test tool is intuitive and easy to learn. For the user with basic knowledge regarding Linux/Unix and good knowledge about the GSM AXE system it will require not more than 1 hour to learn.

6.3 Result of test case executions
The test tool has been executing IT environment test cases continuously every night at 00:30 AM during four weeks. During these tests there have been several incidents where the test tool has discovered failed HTE rigs.

There have been five different types of problems:

- Login problem – The test system could not login to two servers in two different HTE rigs.
  The SSH server application on the Linux servers had failed to start properly.
• Overloaded power supply – The test system could not connect to two servers on one HTE rig. The fuse was overloaded and several servers had failed.

• Operating system failure - The server has stopped answering. It is not possible to contact the server even though the network is functioning properly.

• Daemon missing – The servers have been missing a daemon.

• The Transparent Inter-Process Communication (TIPC) protocol was missing. The TIPC protocol is used for communication between the simulated CP and the RP.
7 Conclusions and future work
This chapter contains the conclusion and future work.

7.1 Conclusions
The following questions were asked in the beginning of the report:

- Which test method is suitable for testing the Hybrid Test Environment?
  Keyword-driven test has been implemented in the test bench. Regression and sanity tests have increased the stability for the HTE rigs.

- Can the test method be automated in the current IT environment?
  The current IT environment is well suited for automated test.

The test tool has been running for four weeks and during this time 7 servers on 4 different HTE rigs have received the verdict “FAIL” in the test tool.
All 7 problems were discovered and fixed before the end users started to use the HTE rigs.
The problems were discovered early and the confidence for HTE was not damaged further.
If the problems can be discovered early and before the end users the reputation and confidence for HTE will improve.

7.2 Future work
The test bench is implemented as a generic solution where many different test cases can be executed. The execution is based on execute and inspect the result. It is possible to continue developing the test bench to test any application that has an interface where a command can be executed and a reply can be retrieved. The test bench application is especially suited for testing the simulator environment in 3G and 4G.

The development of the test tool has focused on the GSM BSC system. Development is ongoing and hybrid test environments will be extended to other areas. If the testing of the HTE system is performed in an early state of the development the stability of HTE will increase and thus the confidence will also increase.
8 Glossary

APG – Adjunct Processor Group
AXE – Automatic Cross-Connection Equipment
BSC – Base Station Controller
BTS – Base Transceiver Station
CORBA - Common Object Request Broker Architecture
CP – Central Processor
GGSN - Gateway GPRS Support Node
GMSC – Gateway Mobile Services Switching Centre
GPRS – General Packet Radio Service
GSM – Global System for Mobile Communications
HTE – Hybrid Test Environment
IPN – Inter Platform Network
MS – Mobile Station
MSC – Mobile Services Switching Centre
PSTN – Public Switched Telephone Network
RP – Regional Processor
RPB – Regional Processor Bus
SEA – Simulated Environment Architecture
SGSN - Serving GPRS Support Node
SPARC - Scalable processor architecture
STE – Simulated Test Environment
SNT – Switching Network Terminal
TIPC - Transparent Inter-Process Communication
TSS – Traffic Simulator System
9 Bibliography


På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Tobias Eiderbrant