Helsinki Metropolia University of Applied Sciences
Degree Programme in Electronics Engineering

Sauli Tuomainen

Test Automation and Continuous Integration on Mobile TV Environment

Final Year Project,
26 May 2009
Instructor: Kari Kärkkäinen, R&D Manager
Supervisor: Janne Mäntykoski, Lecturer

Helsinki Metropolia University of Applied Sciences **Abstract**

Author Title Number of Pages Date	Sauli Tuomainen Test Automation and Continuous Integration on Mobile TV Environment 47 26 May 2009
Degree Programme	Electronics Engineering
Degree	Bachelor of Engineering
Instructor Supervisor	Kari Kärkkäinen, R&D Manager Janne Mäntykoski, Lecturer

This final year project was carried out by commission of Nokia Devices Mobile TV unit. The aim of this project was to design and establish an automatic test environment for a continuous integration process. By automating the testing, it was striven to free resources which were tied to the testing process before. These resources are more valuable to be used for evaluating and developing the other test methods.

At the beginning of this project, the Mobile TV has been described in general. The DVB-H radio network and the other broadcast technologies are introduced as well. Also the prerequisities of the automatic testing, the architecture of the automatic test system and the used test methods, especially on the Mobile TV environment, have been explained.

The actual implementation was started by familiarizing with different testing tools and test environments. After that, the test tools for each testing needs were smoothed out and connected to the automatic test system. Finally, the test system was connected to the Continuous Integration process.

In this project, the Qt based Mobile TV is also described. Qt is an application development framework, which is widely used in the development of graphical user interface applications.

Keywords	Mobile TV, test automation, continuous integration, DVB-H

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Sauli Tuomainen Testiautomaatio ja jatkuva integrointi mobiili TV -ympäristössä 47 sivua 26.5.2009
Koulutusohjelma	elektroniikan koulutusohjelma
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	tutkimus ja kehitys -yksikön johtaja Kari Kärkkäinen lehtori Janne Mäntykoski

Työ tehtiin Nokia Devices Mobile TV -yksikölle. Työn tavoitteena oli suunnitella ja luoda automaattinen mobiili TV -testiympäristö Continuous Integration -prosessiin. Automatisoimalla testausta pyrittiin vapauttamaan testauksen sitomia resursseja, jotka voitaisiin käyttää muunlaiseen testaukseen ja testausvaiheiden kehittämiseen.

Työssä kerrotaan aluksi mobiili TV:stä yleisesti. Siinä kerrotaan DVB-H -radioverkosta sekä muista lähetystekniikoista. Työssä käydään myös läpi automaattisen testauksen edellytykset erityisesti mobiili TV -ympäristössä sekä kuvataan automaattisen testausympäristön rakenne ja käytettävät testausmetodit.

Työssä kerrotaan myös Qt:sta sekä siihen pohjautuvasta mobiili TV:n toiminnasta. Qt on graafinen käyttöliittymäkirjasto ja sillä voidaan kehittää monella eri käyttöjärjestelmällä toimivia ohjelmia tekemättä muutoksia ohjelman lähdekoodiin.

Varsinainen toteutus alkoi tutustumalla erilaisiin automaattisen testauksen työkaluihin ja ympäristöihin. Tämän jälkeen selvitettiin erikseen työkalu kullekin testaustarpeelle ja liitettiin se automaattiseen testausjärjestelmään. Lopuksi testausjärjestelmä liitettiin onnistuneesti Continuous Integration -prosessiin.

Hakusanat	mobiili tv, testiautomaatio, jatkuva integrointi, DVB-H

TABLE OF CONTENTS

Glossary

3GPP	The 3rd Generation Partnership Project. A collaboration between groups of telecommunications associations.
Apache Ant	A software tool for automating software build processes.
API	Application Programming Interface.
ASL	Advanced Scripting Language. A scripting language for user interface testing.
ASTE	Automatic Test System Engine. A system for user interface test automation.
ATLM	Automated Test Lifecycle Methodology
ATS3	Automatic Test System. A system for test automation.
BAM	Broadcast Account Manager.
BSM	Broadcast Service Manager.
C++	A programming language.
CI	Continuous Integration. A software development practice.
CMMB	China Multimedia Mobile Broadcasting. A Chinese broadcasting standard.
DMB	Digital Multimedia Broadcasting. A broadcasting standard.
DVB-H/T	Digital Video Broadcasting-Handheld/Terrestial
ESG	Electronic Service Guide. A digital guide to scheduled broadcast television or radio programs.
FEC	Forward Error Correction. A system of error control for data transmission.
HDTV	High-definition television. A digital television broadcasting with higher resolution than traditional television systems.
IP	Internet Protocol.
ISDB-T	Integrated Services Digital Broadcasting-Terrestial. A broadcasting technology.
MPE	Multi Protocol Encapsulation. A data link layer protocol.
Qt	A cross platform graphical widget toolkit for application development.
S60	A software platform for mobile phones that runs on Symbian.

SCRUM	A software development method.
SIS	Symbian Installation System. An installation file for mobile phones.
STIF	A name of the tool. A testing interface for non-ui components.
TS	Transport Stream.
UHF	Ultra High Frequency.
UI	User Interface.
VHF	Very High Frequency.
XML	Extensible Markup Language.

1. Introduction

Today over 2 billion mobile phones are in use worldwide. The range of services offered on the mobile networks is wide. Mobile TV is one of the newest addition to services supply. It is predicted that the Mobile TV service will reach 200 million users by the year 2011.

Mobile TV software must be tested very carefully to secure functionality between different interfaces, standards and devices. Also performance, power consumption and memory issues must be tested. When technology becomes more and more sophisticated, in Mobile TV software testing there are many things to be taken into account. Because deeply software testing takes plenty of time and resources, there are many helpful tools today available to make the testing process easier.

This thesis describes software testing and software development on the Mobile TV environment. The tools to automate parts of testing are introduced. As well as ways how to make the development work more efficient.

2. Mobile TV Service in General

2.1 Start of the Mobile TV

Mobile TV is a television service for mobile devices, such as mobile phones. There are many variants of Mobile TV network solutions in the world. In May 2005, South Korea became the first country in the world to have a commercial Mobile TV network when it started commercial satellite S-DMB and terrestrial T-DMB services. In Europe, a commercial Mobile TV service was launched in 2006 when German mobile operator MFD established a DMB-based service and an Italian mobile operator 3 established a Mobile TV network based on DVB-H technology. In Finland, a commercial DVB-H based Mobile TV network was launched in December 2006 by Digita.[1;2]

2.2 Mobile TV in Finland

In Finland, Mobile TV service uses DVB-H radio network. In 2006, Nokia and Digita made a contract to provide the world’s first commercial DVB-H-Mobile TV platform supply. The network was established on 1th of December 2006.

In 2009, the Mobile TV service covers the metropolitan area (Helsinki, Vantaa, Espoo, Kauniainen), Turku, Tampere, Oulu and the centre of Salo, coverage is almost 40 percent of people living in Finland. Channel packade D is for DVB-H network and it includes 7 channels: four television channels, Yle1, Yle2, MTV3, The Voice and three radio channels, Iskelmä, The Voice and Radio Nova.[3;4]

2.3 DVB-H Radio Technology

Digital Video Broadcasting – Handheld (DVB-H) is a leading technology standard for the transmission of digital TV handheld devices. This technology was developed as an extension of Digital Video Broadcasting – Terrestial (DVB-T) network with certain additional features and some backwards compatibility. For example, it can share the same multiplex with DVB-T, which makes it suitable for delivery to mobile devices. The first trial of DVB-H service was in 2005. In March 2008, DVB-H was officially endorsed by the European Union as the "preferred technology for terrestrial mobile broadcasting". [5, p. 152;6.]

DVB-H is designed to work on 170MHz-230Mhz, 470MHz-862MHz and 1.452 GHz-1.492GHz bands. DVB-H uses multi-protocol encapsulation (MPE) and forward error correction (FEC). With MPE it is possible to carry packet oriented protocols (for example IP) on top of MPEG-2 Transport Stream (TS). The FEC technology allows the receiver to detect and correct errors without need to ask the additional data from sender.That increases robustness and thus the mobility to the signal. DVB-T provides 2k and 8k modulations and in addition a 4k mode is added to DVB-H. (6, p.219-220;7)

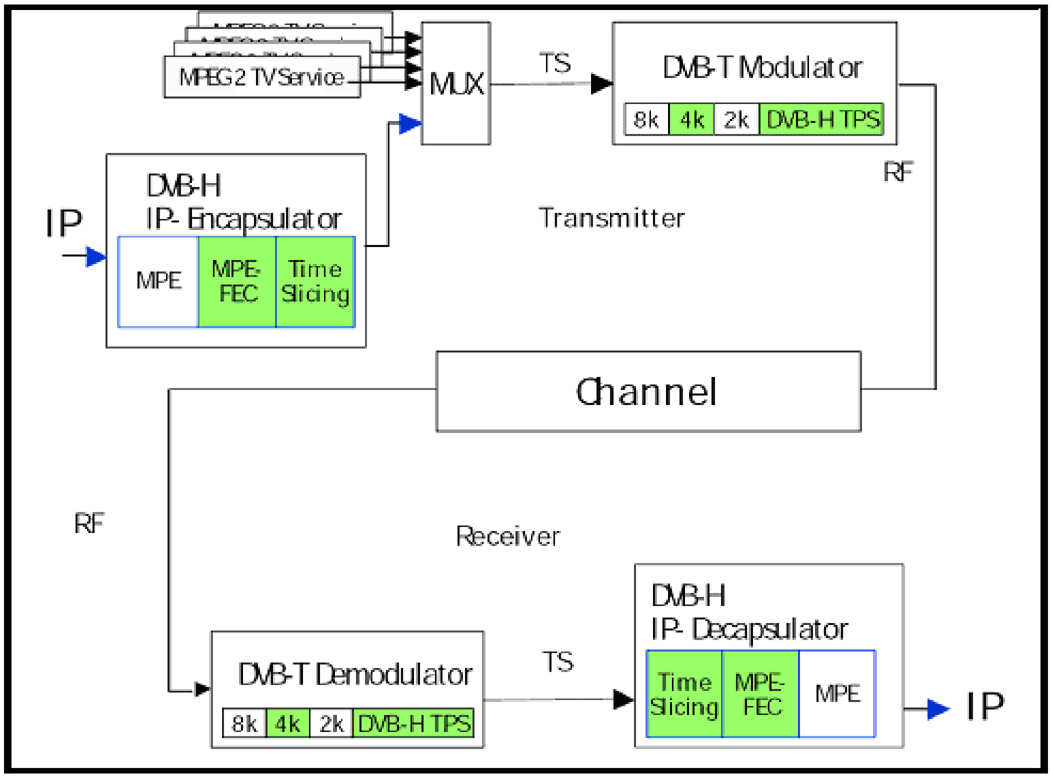


Figure 1. DVB-H Radio Network Structure [8]

Another essential element of DVB-H is the Time Slicing technique. This technique is used to reduce the power consumption of the handheld device. Each TV service in a DVB-H signal is transmitted in burst allowing the receiver to go into the idle mode between bursts when no data is received. The burst bitrate should be at least 10 times higher than the constant bit rate. The time slicing technique reduces power consumption by 90% when compared to the data transmission using the DVB-T technique.
[6, p.223-224;8]

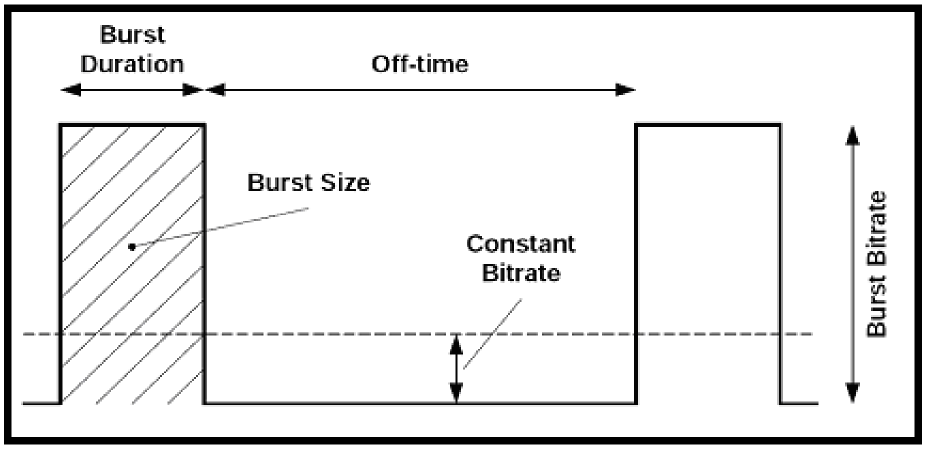


Figure 2. Burst Parameters [8]

2.4 Application Layer

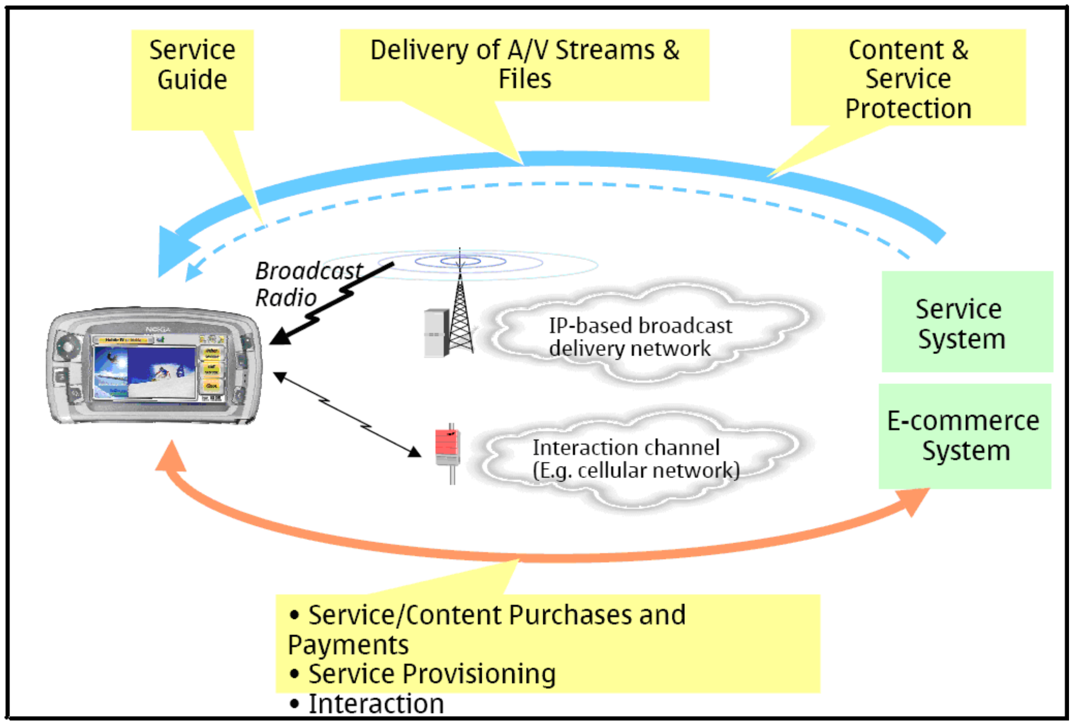

Service Guide
Delivery of A/V Streams & Files
Content & Service Protection
Broadcast Radio
IP-based broadcast delivery network
Service System
E-commerce System
Interaction channel (E.g. cellular network)
• Service/Content Purchases and Payments
• Service Provisioning
• Interaction

Figure 3. Structure of the Application Layer [Based on OMA Mobile Broadcast Services. Toni Paila. Nokia, 2005]

Broadcast network is based on the open standards and therefore creates interoperability among operators and mobile devices. In DVB-H this standard is known as the Open Mobile Alliance Mobile Broadcast Services Enabler Suite (OMA BCAST) and it is an open global specification for a Mobile TV. OMA BCAST is designed to support broadcast technologies such as DVB-H, 3GPP and a mobile unicast streaming system.[14]

The OMA BCAST standard specifies a variety of features including:

Service Guide

Describes all aspects of the service, information about how to purchase service, technical information for terminal how to access to service. Electronic service guide (ESG) supports the pay-per-view feature and also the interactive links. A single ESG can be used by multiple operators. Operators can also provide data files in ESG, for example applications and video clips.

File and A/V stream distribution

The protocols for live streaming and file broadcasting from point to multipoint. It defines how much capacity is reserved for each content and how the content will situate to data caroucel.

Content and Service Protection

OMA BCAST supports open standards based on DRM and the Smart Card profiles. OMA DRM can be applied for many purposes: pay-TV, content protection, ID card, credit card, and so forth. Smartcard profile integrates security software into the SIM card itself.

Interactive Services and Service/Content Purchases

Interactive Services makes it possible that user can participate and access to different services during using Mobile TV. For example, voting during live stream, betting or chatting. When user purhaces a protected content the e-commerce system handles purchase operation and manages rights. [6, p. 236-237]

2.5 Other Mobile TV Broadcast Technologies

CMMB (China)

China Multimedia Mobile Broadcasting standard (CMMB) is used in China and it is intended for use on small screen devices and it resembles a satellite version of DVB-H. It exploits terrestrial transmitters and also satellites, thus the network has a better coverage. It specifies usage of the 2.6GHz frequency band and occupies 25MHz bandwith within which it provides 25 video and 30 radio channels with some additional data channels.[10]

DMB (Korea & Japan)

Mobile TV services delivered directly via satellite are receivable in the areas of the footprint of satellite when users are in an open area. The satellite is a high-powered satellite with transmission in the S-band of 2.630 to 2.655GHz. Inside buildings they use S-band repeaters by which the signals are rebroadcast terrestrially to enable reception. The terrestrial DMB technology uses radio frequency bands III (VHF) and L (UHF) for transmissions and it uses bandwith reduction for the device's power-saving.[6, p. 149-150;11]

ISDB-T /OneSeg (Japan & Brazil)

Integrated Services Digital Broadcasting-Terrestial (ISDB-T) uses 470MHz-770MHz, bandwith of 300 MHz, allocate 50 channels. ISDB-T is designed so that each channel is divided into 13 segments, with a further segment separating it from the next channel. An HDTV broadcast signal occupies 12 segments, leaving the remaining (13th) segment for mobile services. [6, p.165-166;393-338]

MediaFLO (USA)

Media Forward Link Only (MediaFLO) uses 700MHz band but it can operate at any frequency between 300MHz and 1.5GHz. In MediaFLO technology, the receiver can access only that part of the signal which contains the channel being viewed. This reduces devices power consumption and makes it possible to switch channels in switching times of less than 2 sec.[6 p. 161-163;303-304]

3. Software Testing

To ensure that customers get stable and reliable application, the software must be tested. Unstable software may cause lots of additional costs on the market. A primary purpose for testing is to detect software failures and bugs in the source code. The tests must be effective to make the best possible profit. A good testset covers all possible functions in all possible situations. The testsets must also be easy to maintain. The software can be tested at many levels and the testcoverage can be measured in many ways.

There are many tools to make testing easier, to automate testing and to measure testing coverage. Software testing tools are divided into two categories, static analysis tools and dynamic analysis tools. The purpose of static analysis tools is to find errors in the code without any kind of source code executing. A compiler which compiles the code is an example of the static analysis tools. It finds the possible errors while compiling and stops compiling if the error is critical. Dynamic analysis tools test the software while executing the source code.[12]

The testcovegare describes the degree to which the source code of an application has been tested and can be measured at several levels. The source code must be instrumented before the testcoverage can be measured. After instrumenting the tests are run and the tool begins to measure the coverage.

There are a number of coverage criteria, the most common are:

- Functional coverage. How many functions have been executed in the application?
- Statement coverage. Has each line of the source code been executed?
- Decision coverage. How many of the statements have been evaluated to be true and false?
- Entry/exit coverage. Has every possible calls and returns of the function been executed?

With the testcoverage tool, it is also possible to measure the performance of the functions, function call pairs, tasks and timings between any of two given events.[13]

4. Mobile TV Software and Development

4.1 Mobile TV Application Architecture

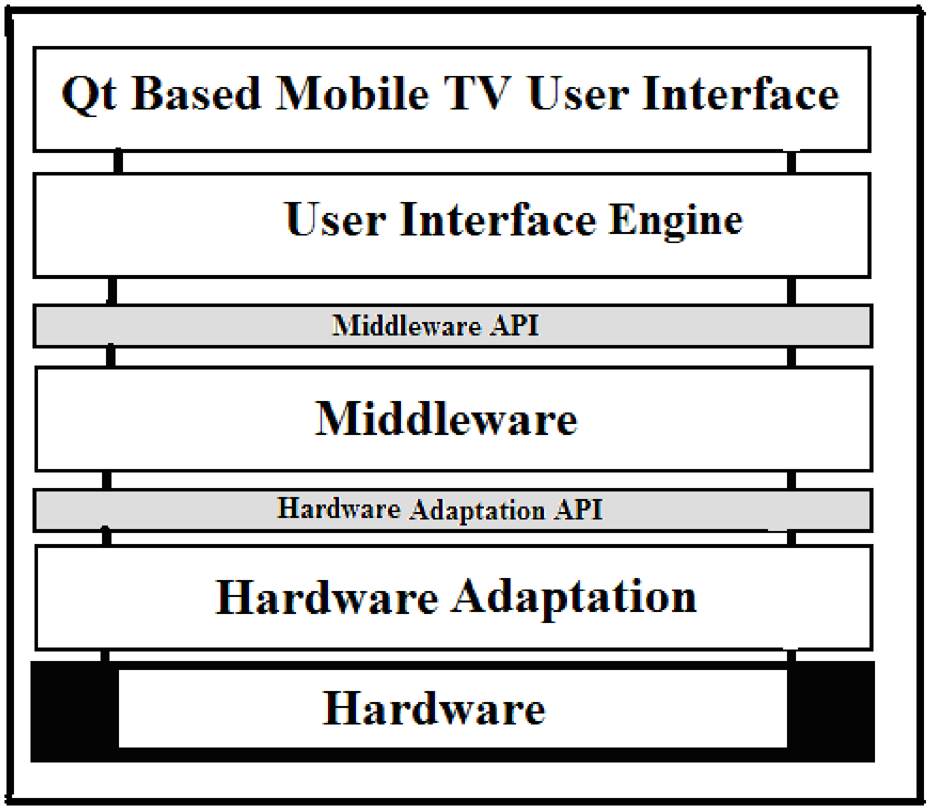


Figure 4. **Architecture of the Mobile TV**

In the figure 4, there are architecture of the Mobile TV, each layer is developed on top of lower layer. Every layer can be tested separately with different tools.

Hardware Layer

The physical hardware layer includes a DVB-H receiver, an antenna and other possible hardware which is needed for receiving data.

Hardware Layer Adaptation

There are drivers for the hardware layer's components in the hardware layer adaption. The layer takes care of the physical components' functionality and it controls the hardware layer.

Hardware Adaptation API

The hardware adaptation API is an Application Programming Interface for the hardware layer. Through the adaption API, for example the middleware's engines and User Interface (UI) can get the "access" to the hardware layer. Therefore, it is possible to control and monitor the basic functionality of the receiver.

Middleware

The middleware layer is the core component of the software platform of a Mobile TV. The middleware consists of different separate modules. The majority of all the functionalities related to the Mobile TV are implemented in this part. Middleware testing can be executed with unittests, every function is called separately and test checks that the right value is returned.

Middleware API

The middleware API makes it possible to use and control the middleware's modules through the user interface. API also provides information concerning traffic between the user interface and the middleware. Middleware API testing can be tested for example with user actions, that user interface return and shows right values on the screen.

User Interface Engine

The User interface provides a visible part of the Mobile TV. The interface shows all the specified information and data on the screen. User interface testing can be executed to check that user interface behaves correctly when some of the keys are pressed.

4.2 Qt-Based Mobile TV

Qt is a cross-platform application development framework provided by Trolltech. With Qt, it is possible to develop applications that run natively on many various platforms; Windows, Linux/Unix, Mac OS X without making any source code changes. Qt is based on the programming language C++, with several non-standard extensions implemented by an additional pre-processor that generates standard C++ code before compilation. Although Qt has been mainly designed for user interface development but it is possible to use Qt when developing pure text based applications.

The Qt-based Mobile TV is developed by using the Qt framework ported to s60. It uses the same middleware engines as the s60 Mobile TV with a “wrapper” between the user interface and the middleware. The user interface has been developed with special graphics libraries which are based on the Qt framework. Thus, in practice, the Qt-based Mobile TV can be used and compiled on many different platforms in the future.

4.3 Mobile TV Development Process

The development of the Mobile TV follows the SCRUM process (see figure 5). The ensemble of the project has been described in the product backlog. It contains backlog items: broad descriptions of all required features, wish-list items, etc. prioritized by business value. For each releasing cycle, “sprint”, the SCRUM teams get backlog items as tasks how to implement software, these items constitute a sprint backlog for each sprint. One sprint lasts typically from two to three weeks. During the sprint, the software is developed and tested based on sprint backlog. The releasing and deeper release testing takes place mainly at end of the sprint.

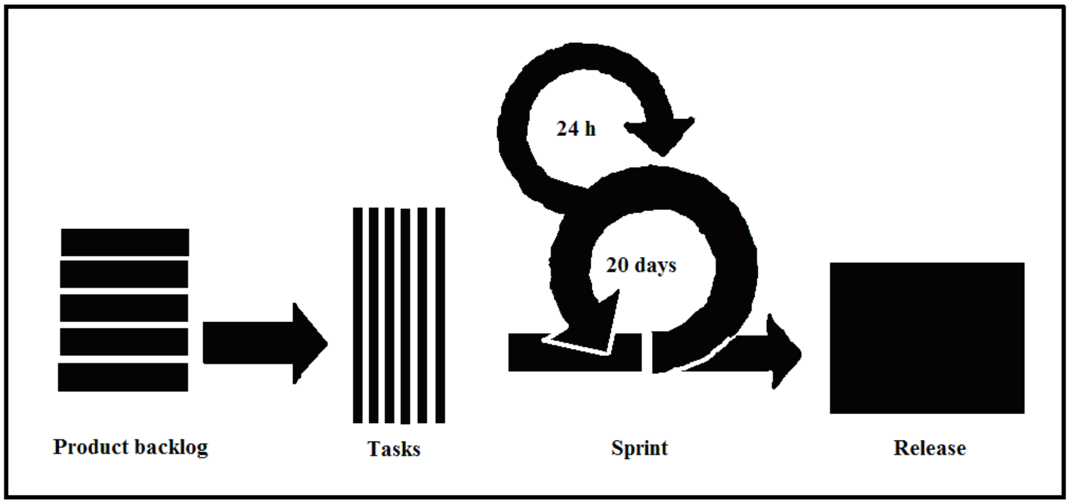


Figure 5. The SCRUM process

If testing is not done continuously, there is a delay to find the possible regression. In addition to this, the deeper testing is done manually. The manual testing ties resources, which could be more valuable to be used for developing the test methods and more specific testing, for example performance testing.

There are many challenges in testing the Mobile TV because it has many dependencies and preconditions which have to be specified before the actual software can be tested. Secondly, there are many different standards which are not compatible with each other. Therefore, there are many things which have to be taken into account when testing the Mobile TV. This also takes lots of time because every possible environment variable and situation has to be simulated.

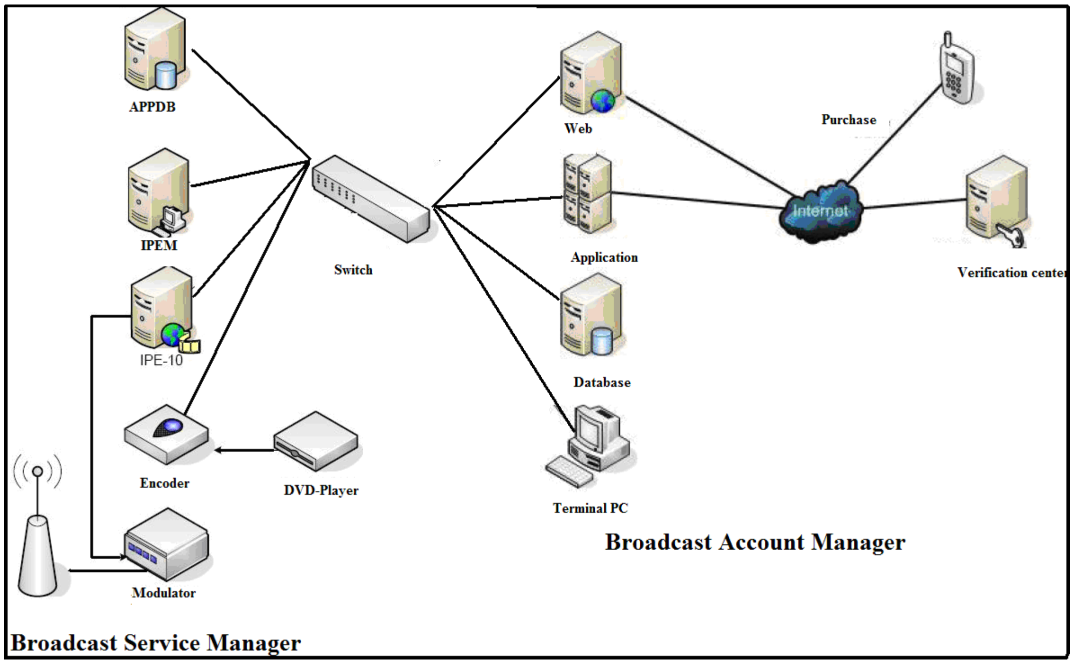


Figure 6. The Architecture of the Test Environment.

With the test environment it is possible to add content to the DVB-H network. This makes it possible to simulate different conditions and software can be tested in many different situations and environments. The test environment consists of two main parts: Broadcast Service Manager (BSM) and Broadcast Account Manager (BAM). These both are controlled via PC.

Broadcast Account Manager is a service fulfillment and charching solution. With BAM, it is possible to simulate different purchase situations. It handles purchase events and charging when chargeable content is bought. Requests and rights are routed via the cellurar packet IP services.

Broadcast Service Manager makes it possible to add channels and programs to test network, in other words, it is possible to specify the content of ESG. The BSM database manages all the service databases and profiles. The IPE elements broadcast the given streams into the designated DVB-H time slices. The IPE manager enables central management if there are many IPEs.

The content of the stream can be received, for example from a DVD player and an encoder converts the digital data into a standard TV signal and it encrypts content if needed. Finally, the wanted content is sent to DVB-H network via an antenna.

5. Continuous Integration

The enhancement of the Mobile TV development was started by making the release cycle more frequent and by automating the testing. This made it possible to detect errors and integration problems and make fixes continuously. Thus in theory, software can be released at any time quite fast. This process is called a continuous integration.

The software projects involve lots of source code files that need to be orchestrated together to build a release. When there are many people involved, keeping track of all of these source code files is a major effort. There are many different codebase management tools. With Source Control Management (SCM) tool, it is possible to avoid problems when developers may be changing the same files, for example an initial set of files is "version 1" when the first change is made to file, the resulting file is "version 2". This makes possible to get older version of files. Secondly, when there are many delelopers working with the codebase and the code changes frequently, it is good to know the possible codebreaks and conflicts as soon as possible. Codebreaks that stay undetected for weeks can be very hard to resolve. The whole point of continuous integration (CI) is to provide rapid feedback.[15]

Continuous integration is a development mode which has many advantages:

- Immediate testing for changes, early warnings of broken code
- Always working code, developer can revert the codebase back to a error free-state
- Development is more iterative and effective
- Monitoring test- and build results in one place.

The continuous integration system has no fixed recipe. The system setup can depend on the size and needs of the development team. So, there are many different tools available to control, automate and monitor the software development.

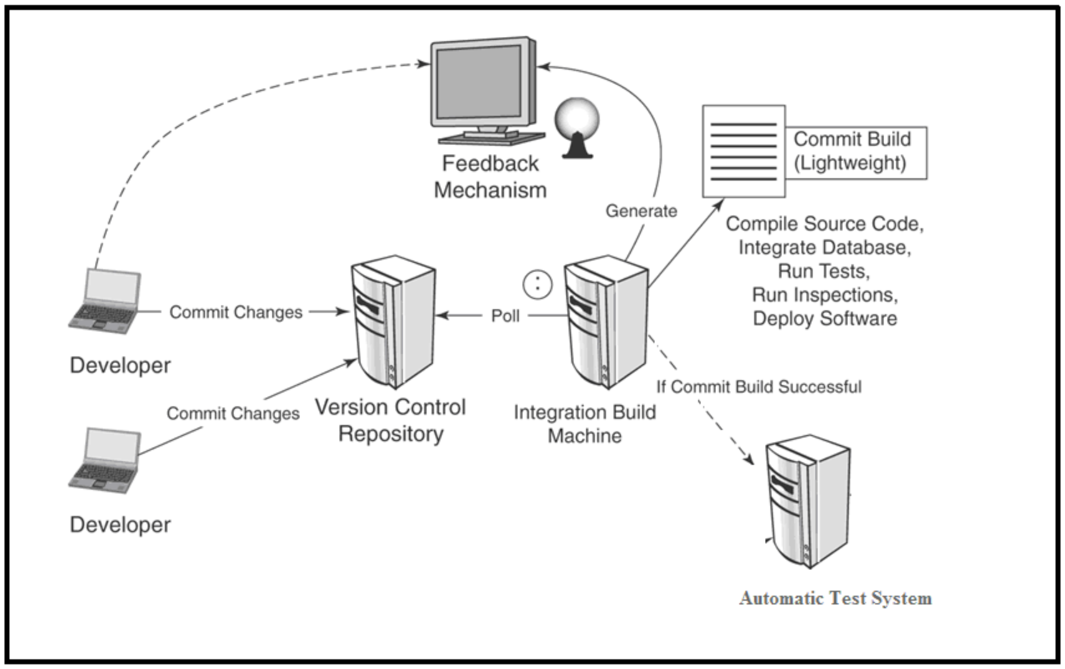

Feedback Mechanism
Generate
Commit Build (Lightweight)
Compile Source Code, Integrate Database, Run Tests, Run Inspections, Deploy Software
Developer
Commit Changes
Poll
Commit Changes
Version Control Repository
Integration Build Machine
If Commit Build Successful
Developer
Automatic Test System

Figure 7. **Continuous Integration setup [Based on (15)]**

This kind of development mode gave a solution to make the Mobile TV development more effective:

- Developers check in changes they have made to the codebase management system and they can get the latest "working" code from the version control repository

- The building system is integrated with the version control repository and it compiles the source code from the database and makes a release candidate. The release cycle and building platform can be controlled. In addition, the built system also compiles the preselected test sets

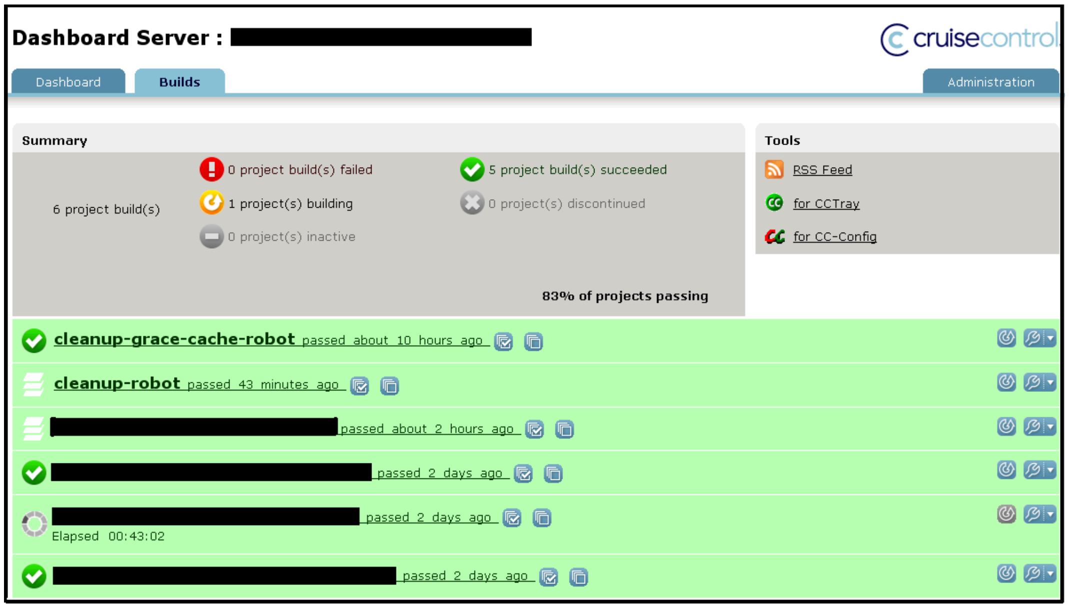
- The release candidate and tests are sent to an automatic test system. The test system executes test cases for the release candidate and collects the results.

5.1 Building and Cruise Control Dashboard

The build framework is controlled by Cruise Control, which is a Java written continuous integration tool and-uses XML files for configuration. Cruise Control controls the build system and visualizes the project's build status with Cruise Control Dashboard. Dashboard is a web interface, which provides details of the current and previous builds.

The build system is based on open source Apache Ant, which is a software tool for automatic software build processes. Ant is quite similar to the commonly used build tool Make, but Ant uses XML files to describe build process and it is dependencies, whereas Make uses Makefile format.

Cruise Control monitors version control repository every half an hour and if there is a new change to code, "task", checked in, Cruise Control triggers the building process. The build process can also be scheduled so example building environment updating takes place during a nighttime. The build system compiles the source code into binaries and makes a release candidate. The build results and logs can be seen in Cruise Control Dashboard.



Dashboard Server : [redacted]

cruisecontrol

Dashboard | Builds | Administration

Summary

6 project build(s)

0 project build(s) failed

1 project(s) building

0 project(s) inactive

5 project build(s) succeeded

0 project(s) discontinued

83% of projects passing

Tools

RSS Feed

for CCTray

for CC-Config

cleanup-grace-cache-robot passed about 10 hours ago

cleanup-robot passed 43 minutes ago

[redacted] passed about 2 hours ago

[redacted] passed 2 days ago

[redacted] passed 2 days ago
Elapsed 00:43:02

[redacted] passed 2 days ago

Figure 8. Cruise Control Dashboard

Cruise Control Dashboard shows information on the building processes, status, durations and elapsed times. In figure above, there are three different projects. Via dashboard, it is also possible to force the building process to start again and setup building parameters.

Symbols indicate the status of the projects:

🟩 Indicates that project build has been successful for less than 24 hours.

◌ Indicates that the project passed to the building process for the last time and is currently building.

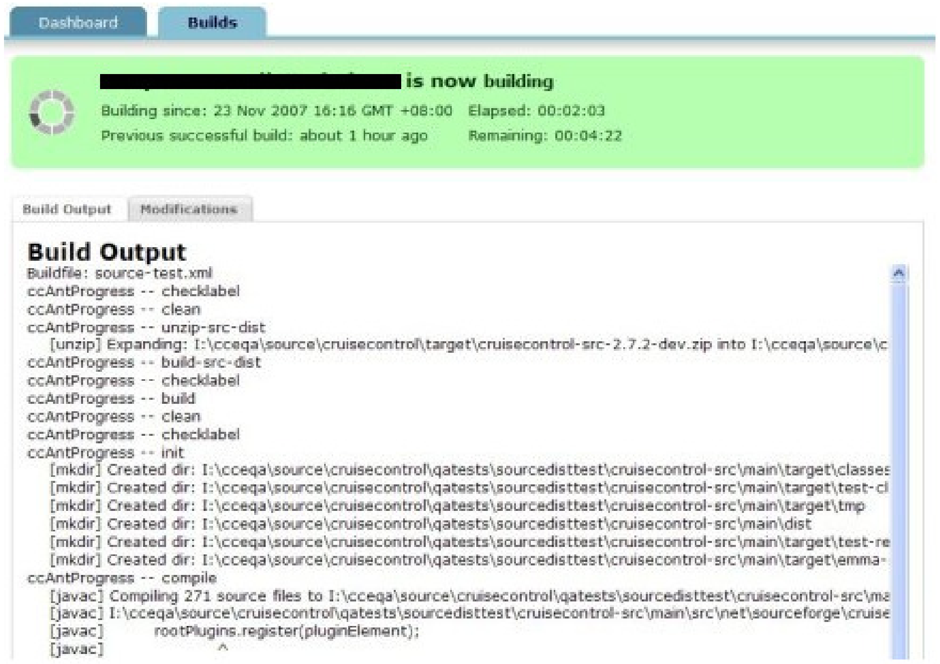
⬜ Indicates an inactive project.

🟥 Indicates a project build that has failed for more than 24 hours.

There are couple of other symbols used in the dashboard:

Indicates the project build has been successful for more than 24 hours and is currently in the queue waiting to the next build.

Indicates a project build that has failed for less than 24 hours and is currently paused.



[redacted] is now building

Building since: 23 Nov 2007 16:16 GMT +08:00   Elapsed: 00:02:03
Previous successful build: about 1 hour ago   Remaining: 00:04:22

Build Output

```
Buildfile: source-test.xml
ccAntProgress -- checklabel
ccAntProgress -- clean
ccAntProgress -- unzip-src-dist
   [unzip] Expanding: I:\cceqa\source\cruisecontrol\target\cruisecontrol-src-2.7.2-dev.zip into I:\cceqa\source\c
ccAntProgress -- build-src-dist
ccAntProgress -- checklabel
ccAntProgress -- build
ccAntProgress -- clean
ccAntProgress -- checklabel
ccAntProgress -- init
   [mkdir] Created dir: I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\target\classes
   [mkdir] Created dir: I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\target\test-cl
   [mkdir] Created dir: I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\target\tmp
   [mkdir] Created dir: I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\dist
   [mkdir] Created dir: I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\target\test-re
   [mkdir] Created dir: I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\target\emma-
ccAntProgress -- compile
   [javac] Compiling 271 source files to I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\ma
   [javac] I:\cceqa\source\cruisecontrol\qatests\sourcedisttest\cruisecontrol-src\main\src\net\sourceforge\cruise
   [javac]        rootPlugins.register(pluginElement);
   [javac]              ^
```

Figure 9. Build logs

In the figure 9 above is the build detail page for a project that is currently building. It is possible to see build logs while the project is building, the build output is displayed on-screen. If error occurred and resulting in a build break, errors can be seen from the logs.

5.2 Automatic Testing

Software test automation makes testing more effective. It means that computers run already existing manual tests automatically and collect the results. Before it is possible to automate testing, there must be a working manual testing environment. The manual testing environment must contain test conditions and expected results, and the environment must be standalone and it should be possible to restore the default settings if an error occurs. Standalone environment enables repeatability of the testruns. Without these pre-conditions automatic test tools have no advantage.

Automatic testing makes easier to find possible regression, if same tests are run frequently. One common expedient to automate testing is to use scripting. It is series of commands that is run one after the other. With scripting it is also possible to set automatically test conditions and presteps before the actual testing is run.

Before starting to automate testing, it should be to investigated which parts of the testing could automated. It is not possible to automate all the testing process, sometimes computers do not find all the errors which can be found by human, for example minor graphics errors from user interface. Ad hoc testing typically exposes the most of the errors and bugs. But for example when there is a long sequence of number computer finds possible errors from sequence easier.

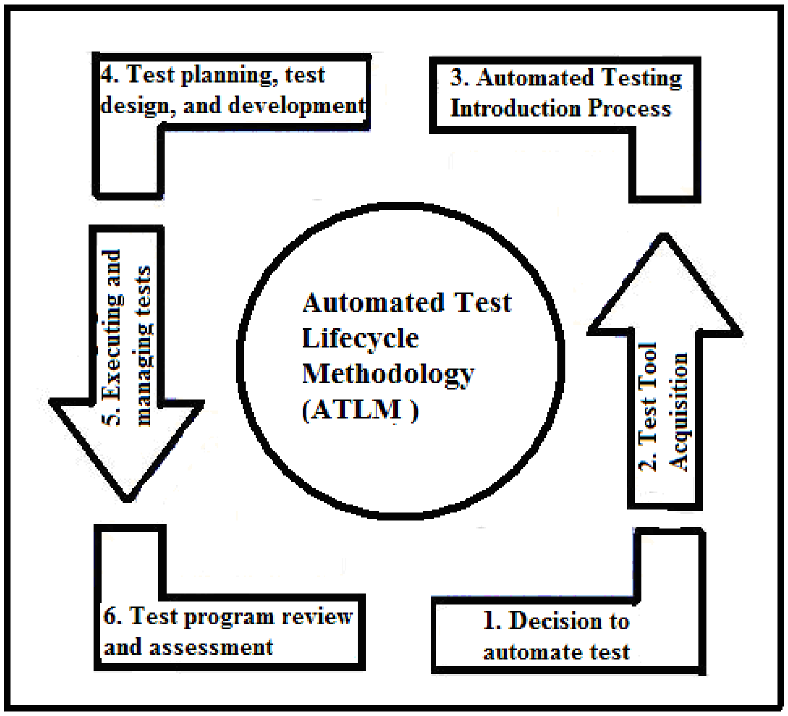

4. Test planning, test design, and development
3. Automated Testing Introduction Process
5. Executing and managing tests
Automated Test Lifecycle Methodology (ATLM)
2. Test Tool Acquisition
6. Test program review and assessment
1. Decision to automate test

Figure 10. Automated Test Lifecycle Methodology [16]

Elfriede Dustin introduces in his book [16] Automated Test Lifecycle Methodology (ATLM). According to him, lifecycle includes 6 main processes:

1. Decision to automate tests:

- When and what to automate
- Evaluating estimated return on investment
- Managing expectations
- Value of test automation.

2. Test Tool Acquisition:

- Knowing which tools are available and compatible with your technology, understanding tool needs, and then selecting the compatible test tool
- List of types of tools available, tool evaluation and recommendations.

3. Automated Testing Introduction Process:

- What is involved in introducing test automation
- Things to take into account and lessons learned
- Training considerations.

4. Test planning, test design, and development:

- Test automation standards. Design and development standards
- Test automation requirements/design/develop/test automation
- Test automation frameworks.

5. Executing and managing tests:

- Maintaining the automated tests
- If new defects are uncovered, add test scenarios to the automation framework
- Deliverables from various testing phases.

6. Test program review and assessment:

- Evaluate the outcome, constant evaluation
- Evaluate improvement opportunities.

5.3 Automatic Test System used for Mobile TV Testing

After a successful building process, the release candidate is made and it is sent to the Automatic Test System (ATS). The building framework compiles also all wanted tests, which are configured in the building system's xml configuration files.

There are different test frameworks and tools available for different testing purposes. ASTE, QTestlib and STIF are tools and frameworks for executing and implementing the test cases. For Mobile TV, we decided to use several test framework setups and combinations for UI and middleware testing.

Qt based Mobile TV wrapper testing takes place with QTestlib. Middleware testing takes place with STIF framework and user interface testing happens with ASTE framework.

ATS is a Java -based application. It provides a WEB user interface for executing tests and managing test plans. Test execution is controlled via an XML file and all the executable tests are in one package, in the testdrop. The testdrop package includes all the needed files for testing; flash test images, installation packages, test libraries, testcases and the xml file to control the test run. ATS flashes the testdevice and copies all needed files to the right directories in the phone. In addition, ATS collects test results and shows them on the user interface or/and sends the results to the configured email address.

The system contains an ATS server and zero or more ATS clients. The clients can be installed also in the same PC with an ATS server if needed. These clients will form the ATS peer-to-peer network. Each client can contain a number of devices and different drivers can be registered for single hardware so the number of test framework drivers, in this case ASTE and STIF can be used on the same test device. One big advantage of ATS is that it is possible to setup test system with number of different devices and run tests parallel on different types of devices. This reduces time consuming of the test run when different tests can be run at the same time.

The drivers are configured using the ATS configuration file. In this file, every driver's properties are configured, for example connection type and how long ATS waits after reboot.

The STIF driver enables execution of test cases made on STIF framework.
The ASTE device driver enables running of ASTE keywords.
The EUNIT driver is for executing Qt written unit tests for Qt based Mobile TV.

Every testdrop package includes test.xml, it is usually made automatically. Xml file tells properties of the testrun to ATS;

```
<target>
   <device rank="none" alias="Device_rank">
     <property name="TYPE" value=" Device_type "/>
     <property name="HARDWARE" value=" Device_hardware"/>
     <property name="HARNESS" value="STIF"/>
   </device>
 </target>
```

In this case HARNESS = STIF and TYPE = Device_type. ATS reads the properties and it finds specified test device which has a STIF driver installed and starts to execute the testset. If device is not found, testrun failed at startup.

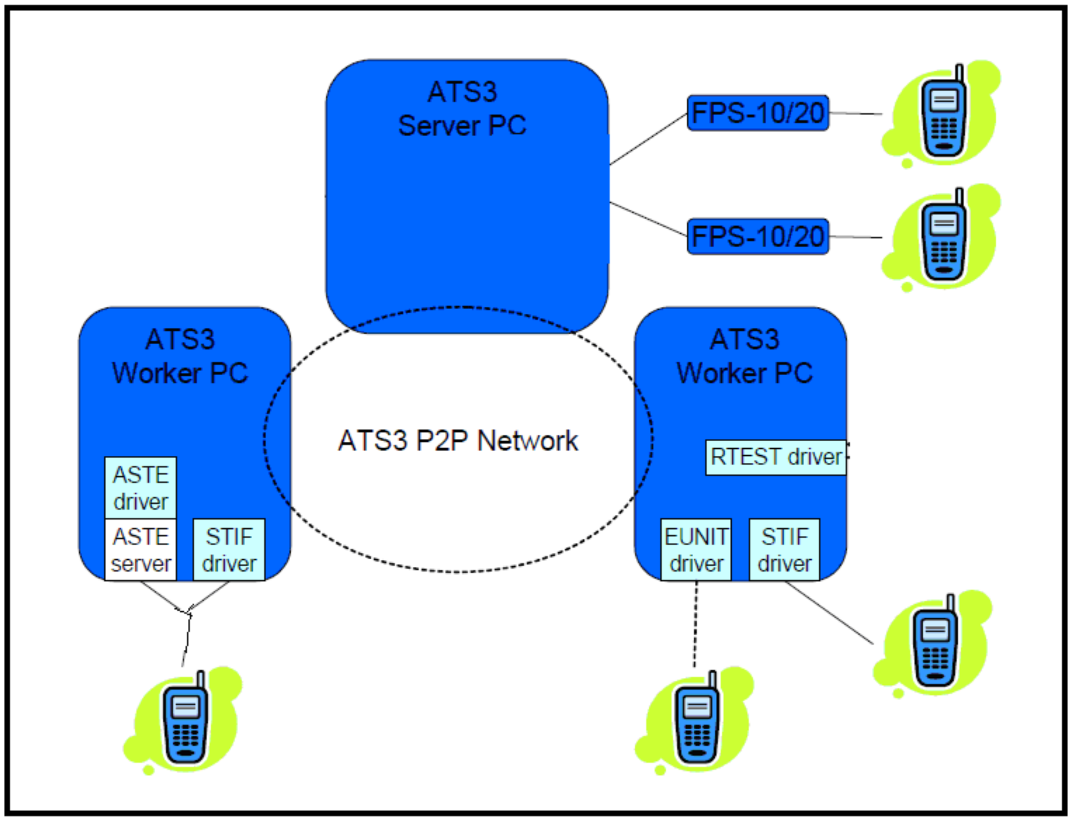
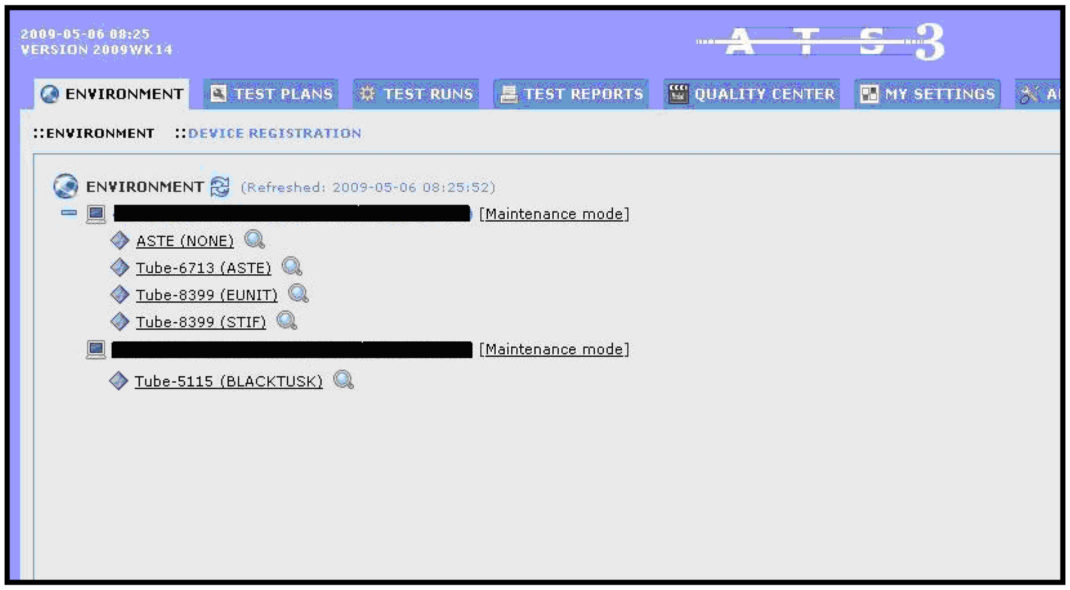


Figure 11. Automatic Test System setup [Based on (18)]

In the figure 11 above, there are ATS setup with two client PCs. The test devices are connected to PC with prommer which handles the flashing. The server PC also contains a database where the testruns are saved.



2009-05-06 08:25
VERSION 2009WK14

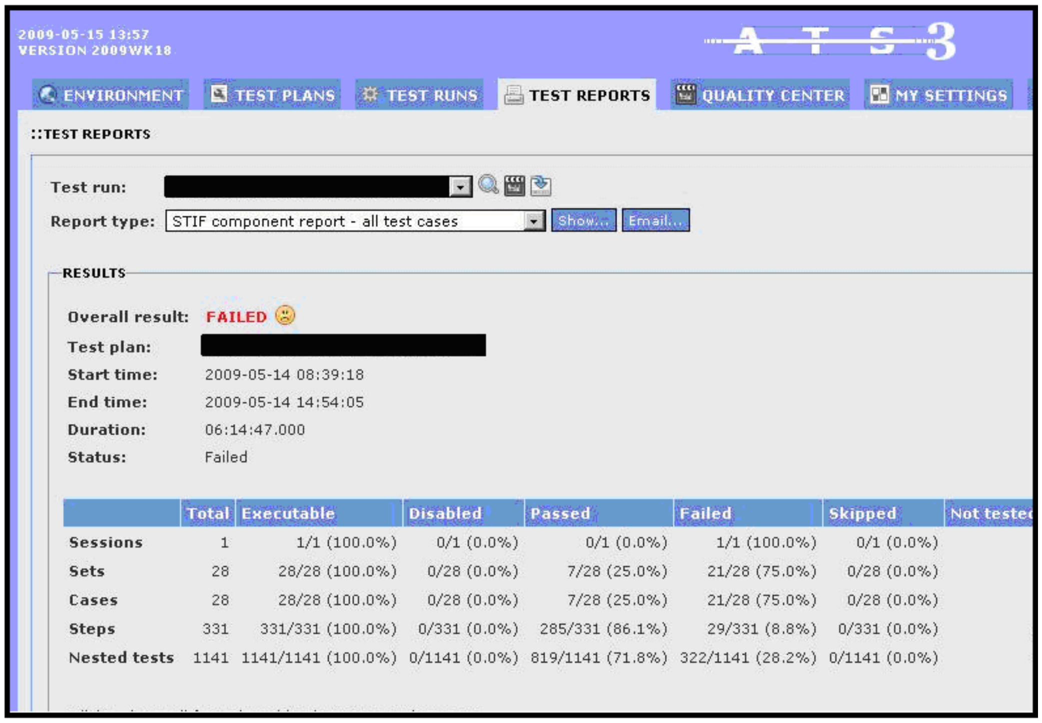
::ENVIRONMENT ::DEVICE REGISTRATION

ENVIRONMENT (Refreshed: 2009-05-06 08:25:52)

- [redacted] [Maintenance mode]
  - ASTE (NONE)
  - Tube-6713 (ASTE)
  - Tube-8399 (EUNIT)
  - Tube-8399 (STIF)
- [redacted] [Maintenance mode]
  - Tube-5115 (BLACKTUSK)

Figure 12. ATS3 setup ASTE,EUNIT and STIF drivers installed.

In the figure 12 above there are test framework drivers installed. This enables test runs with ASTE, EUNIT and STIF frameworks.



2009-05-15 13:57
VERSION 2009WK18

::TEST REPORTS

Test run: [redacted]
Report type: STIF component report - all test cases

RESULTS

Overall result: FAILED
Test plan: [redacted]
Start time: 2009-05-14 08:39:18
End time: 2009-05-14 14:54:05
Duration: 06:14:47.000
Status: Failed

	Total	Executable	Disabled	Passed	Failed	Skipped	Not tested
Sessions	1	1/1 (100.0%)	0/1 (0.0%)	0/1 (0.0%)	1/1 (100.0%)	0/1 (0.0%)	
Sets	28	28/28 (100.0%)	0/28 (0.0%)	7/28 (25.0%)	21/28 (75.0%)	0/28 (0.0%)	
Cases	28	28/28 (100.0%)	0/28 (0.0%)	7/28 (25.0%)	21/28 (75.0%)	0/28 (0.0%)	
Steps	331	331/331 (100.0%)	0/331 (0.0%)	285/331 (86.1%)	29/331 (8.8%)	0/331 (0.0%)	
Nested tests	1141	1141/1141 (100.0%)	0/1141 (0.0%)	819/1141 (71.8%)	322/1141 (28.2%)	0/1141 (0.0%)	

Figure 13. Mobile TV middleware testrun results

In ATS it is possible to see test reports and more details of the testrun. In the figure 12 there are the Mobile TV's middleware test run results. The STIF framework is used and in this run there are 1,141 testcases executed.

5.4 Automatic Test System Framework Drivers

5.4.1 ASTE

Automatic System Test Engine (ASTE) is an UI Test Automation tool. ASTE is a server solution. So there is no web user interface. Every possible end-user action can be simulated with using the ASTE keyword based scripting language. The basic idea in scripts is to do actions on the device, like press a button or select a menu item and then verify that device worked correctly using, for example, the text verification.

The most important components of ASTE are keywords and Test Execution Management (TEM) Tool application. ASTE framework can handle approximately 15 keywords. These keywords are listed to the test file.

TEM is a windows application which controls testing. With TEM, it is possible to add, remove or discontinue the test case execution. It also makes it possible to execute a single test case. The test execution with the ASTE system begins from the testfile execution. ASTE checks that all the necessary test devices are available. If it fails to find devices it explores them as long as the devices are ready. When the devices are found, it starts to execute the keywords.[17]

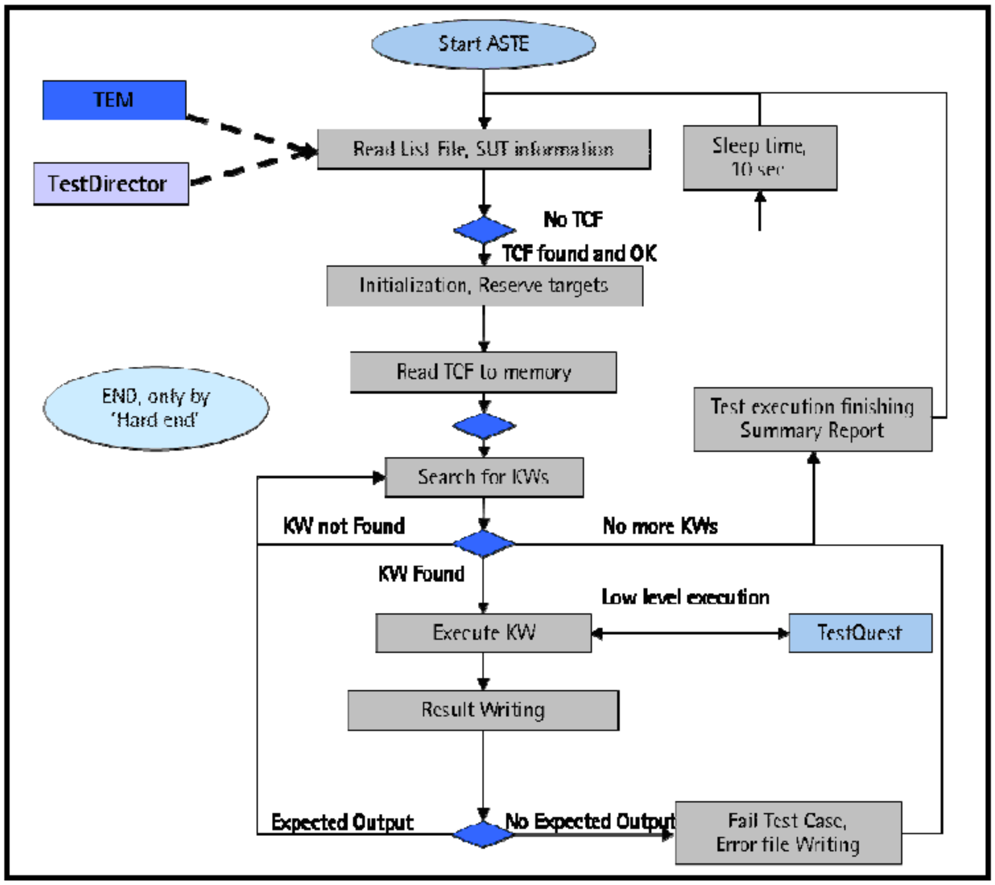


Figure 14. Architecture of the ASTE [17]

In the figure 14 there are ASTE testrun architecture introduced. ASTE finds test case file and when it can be found it initializes and reserves test devices. After testcase file is readed to memory, keywords execution begins. When all the keywords are executed test run results is summarized.

5.4.2 QtTestLib

QtTestLib framework is provided by Trolltech. It is a tool for unit testing Qt -based applications and libraries. There are four functions that are not treated as test functions. They will be executed by the testing framework and can be used to initialize and clean up either the entire test or the current test function.

- initTestCase() will be called before the first test function is executed
- cleanupTestCase() will be called after the last test function was executed
- init() will be called before each test function is executed
- cleanup() will be called after every test function.

QTestlib provides macros to verify testresults, for example, QCOMPARE and QVERIFY:

QCOMPARE (*actual,expected)*

The macro compares an *actual* value to an *expected* value using the equal operator. If *actual* and *expected* are identical, execution continues. If not, a failure is recorded in the test log and the test will not be executed further.

QVERIFY (*condition*)

The macro checks whether the *condition* is true or not. If it is true, the execution continues. If not, a failure is recorded in the test log and the test won't be executed further.

Test cases are compiled with qmake and test results can be collected from the log file and log is saved to testdevice.

5.4.3 STIF

STIF is a sort of testing interface for non-ui components. The STIF test framework is a tool for implementing tests and testing. STIF uses different interfaces for testing: STIF Console UI, STIF series 60 UI and STIF ATS interface. (see figure 15). STIF is installed to a device with installation package and each middleware engine’s test is listed in testcase file.

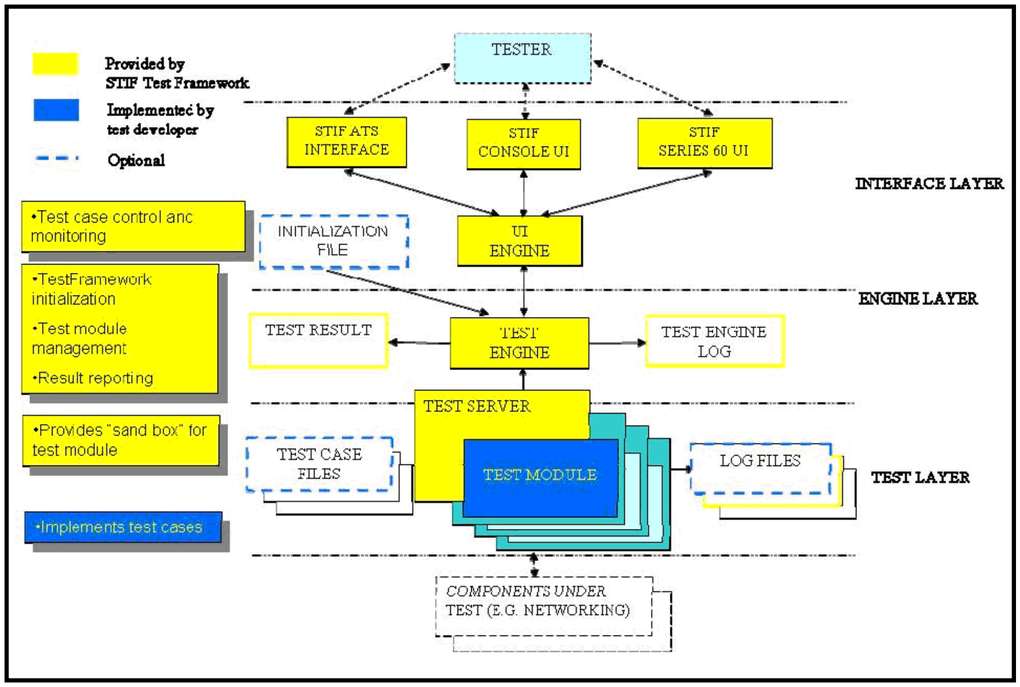


Figure 15. The Architecture of the STIF [19]

6. Middleware Testing

6.1 Mobile TV

STIF test framework is used for middleware testing of the Mobile TV. The test system client has a STIF driver installed and the device is connected to a client. To do so, it is possible to flash images through ATS when needed.

The test cases are specified in the build system configuration files and after building a testdrop has been made. It is also possible to add poststeps and presteps to test.xml. This is done with xml files, which are in the same folder with the test case. So, it is possible to setup pre-conditions before the actual test is executed. In this case, an installation package is installed. The test cases measure the functionality of the middleware and all the modules of the middleware's engine are coveraged.

The testdrop zip file could include flash images, testcases, binaries which have to be tested and the test.xml file which is used to control testruns.

The STIF framework needs to be installed to a device before the test execution can be started. The syntax of the testcase itself is quite simple and the testcases are usually small. For example testcase.cfg with one test could be something like this:

```
[Define]
ReceiverInactive 1
ReceiverActivating 2
ReceiverDisabled 3
[EndDefine]

[Test]
title Mobile TV receiver test

create receiverstate tstreceiver
tstreceiver GetReceiverState

delete tstreceiver
[EndTest]
```

The testcase above tests one middleware's engine. It asks the receiver state, there are no conditions, so this test case passes always if any receiver status is received.

6.2 Qt Based Mobile TV

The same worker is used for the Qt Based Mobile TV middleware testing as for Mobile TV. The Qt Mobile TV middleware testing is executed with unit tests. The testcases are written with the QtTestlib framework. On ATS, unit tests are executed with the EUNIT driver.

The structure of testdrop for Qt Mobile TV middleware testing is quite similar to the STIF testdrop for Mobile TV. The testdrop includes images, installation package and test.xml file for control testrun. The only difference is that each Qt unit test is an application itself and it is built and compiled when creating the testdrop. The unittest executable is installed to the phone and ATS collects the testresults.

It was not possible to make a Qt unittest testdrop on the building system automatically because the build system did not support Qt yet. The solution was to make a testdrop with the Python script language after every building process. The script puts the wanted unit tests and images to testdrop and sends it to ATS server.

The Qt unit test could be something like this: QtTest class and testable header must be included. The test functions are introduced as slots and the test function never has a return value. QCOMPARE macro returns either true or false. If the compare returns false, the test execution stops immeadiately. With the QTEST_MAIN macro, it is possible to compile standalone tests.

```
#include "tested1.h"
#include <QtTest/QtTest>

class Test: public QObject
   {
   Q_OBJECT

private slots:
   void testfunction();
};

void Test::testfunction()
{
test testname;

Qstring name;
name = "test";

QCOMPARE(test.name(),name);
}
QTEST_MAIN(Test)
#include " unit_test.cpp"
```

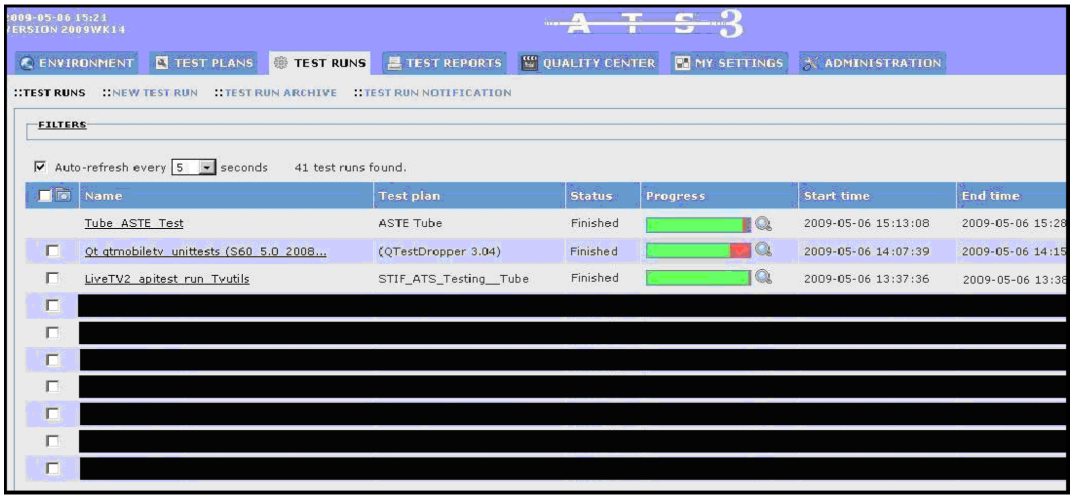
7. User Interface Testing

The User Interface testing is executed with the ASTE driver. The tests are run with the touch screen device, thus UI testing for Mobile TV and Qt based Mobile TV can be executed with same clients. The functionality of the UI is different in Mobile TV than in Qt based Mobile TV, so minor changes must be made to the test code. However, some parts are the same. It is also possible to verify results with bitmap file. Test server compares the picture on the screen with the expected bitmap file; these bitmaps must be included to testdrop as well.

The ASTE test framework driver is on the server PC side as a client. The testdrop creation iscontrolled with xml configuration files, which define all needed parameters; example ATS server address and wanted test framework driver. On ATS side, testrun is controlled via test.xml file. When testing with ASTE it is mandatory to use particular theme on the phone, so basically before every testrun theme is installed to the device.

Here is a basic example of ASTE test. When the device is starting, the date, time and region is set by pressing 3 times a soft left key.

```
kw_AddComment
"Fill in the values for the startup queries, home country, date and time"

kw_Type/"Alpha"        "Master->§ui_device_home_country§"
kw_PressHardkey        <SoftLeft>
kw_WaitTime            3
kw_PressHardkey        <SoftLeft>
kw_WaitTime            3
kw_PressHardkey        <SoftLeft>
kw_WaitTime            10

kw_AddComment          "Set time to server time"
kw_SetTime"%SERVER_TIME%"

kw_Capture"X3"

kw_AddComment
"Press end key some times to close possible welcome application and possible crash
notes"

kw_PressHardkey        <End>/<"End>"/<End>
```



☑ Auto-refresh every 5 seconds   41 test runs found.

☐	Name	Test plan	Status	Progress	Start time	End time
	Tube_ASTE_Test	ASTE Tube	Finished		2009-05-06 15:13:08	2009-05-06 15:28
☐	Qt qtmobiletv_unittests (S60_5.0_2008...	(QTestDropper 3.04)	Finished		2009-05-06 14:07:39	2009-05-06 14:15
☐	LiveTV2_apitest_run_Tvutils	STIF_ATS_Testing__Tube	Finished		2009-05-06 13:37:36	2009-05-06 13:38
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
☐	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Figure 16. ASTE, Qt unit test and STIF testruns

The figure 16 shows the test results for three different testruns. The test execution for Qt -based Mobile TV's middleware (QtTestlib) and user interface (ASTE). One testrun is for Mobile TV's middleware (STIF).

8. Conclusions

This project was part of a Mobile TV development enchancement. The aim was to make the testing process more effective. The attempts to reach this goal were by automating some parts of the test process. The automating was completed by establishing an automatic test system with the ATS framework and by connecting ATS to a part of the continuous integration process.

The goals of this project were reached but there were some limitations to the study. It was not possible to automate the Qt -based Mobile TV and user interface testing. The reasons for this were the limitations in the building system at the moment. Thus, only the test of the Mobile TV's middleware was automated.

As the result of this project, the tests for the Mobile TV's middleware are executed automatically after each build on the ATS system. Over 2,500 testcases are running on ATS. One testround takes four to six hours, depending how many test devices are used. If that big test set are run by manually it takes up to two days. In addition to this, an automatic test system was established. More testing will be automated later when the building framework gets more functionality and support.

After the automatic test environment has been created, it makes it possible to develop the test methods for the Qt -based Mobile TV's middleware and the User Interface testing. The testing will be then more effective and the recources can be used, for example for more specified testing.

List of References

1 Mobiilitelevisio. (WWW-document). Many writers.
http://fi.wikipedia.org/wiki/Mobiilitelevisio Accessed 04 April 2009.

2 Mobile TV. (WWW-document). Many writers.
http://en.wikipedia.org/wiki/Mobile_TV Accessed 04 April 2009.

3 Nokia Provides Broadcast Solution to Digita. (WWW-document). Softpedia.
http://news.softpedia.com/news/Nokia-Provides-Broadcast-Solution-to-Digita-23653.shtml Accessed 07 April 2009.

4 Näkyvyysalueet. (WWW-document). Digita Oy.
http://www.mobiilitv.fi/Nakyvyysalueet Accessed 10 April 2009.

5 Amitabh Kumar. Mobile TV: DVB-H, DMB, 3G Systems and Rich Media Applications. United States of America: Focal Press, 2007

6 Press Release. (WWW-document). EUROPA.
http://europa.eu/rapid/pressReleasesAction.do?reference=IP/08/451
Accessed 12 April 2009.

7 Time Slicing. (WWW-document). Many writers.
http://en.wikipedia.org/wiki/Time_slicing Accessed 27 March 2009

8 Bernd Schloer, DVB-H Time Slicing. Thesis. University of Goettingen, 2007

9 OMA BCAST. (WWW-document). Many writers.
http://en.wikipedia.org/wiki/OMA_BCAST Accessed 02 May 2009.

10 China Multimedia Mobile Broadcasting. (WWW-document). Many writers.
http://en.wikipedia.org/wiki/CMMB. Accessed 20 April 2009.

11 Telecom ABC. (WWW-document). Many writers
http://www.telecomabc.com/d/dmb.html. Accessed 14 April 2009.

12 SEB Foundation Certificate in Software Testing. (WWW-document).
http://www.cs.helsinki.fi/u/taina/ohte/s-2004/luennot/Luku02.pdf Accessed 23 April 2009.

13 Steve Cornett. Code Coverage Analysis. (WWW-document).
http://www.bullseye.com/coverage.html Accessed 09.05.2009.

14 The OMA BCAST Standard for Bearer Independent Mobile TV Services. (WWW-document). Ericsson Research.
www.lkn.ei.tum.de/~steinb/ACMMV2007/content/slides/OMA_BCAST.pdf
Accessed 21 April 2009.

15 Paul Duvall. Continuous Integration: Improving Software Quality and Reducing Risk. United States of America, Indiana: Pearson Education, 2007

16 Dustin, E., Paul, J., Rashka, J. Automated Software Testing: Introduction, Management and Performance. 6th edition. United States of America, Boston: Addison-Wesley, 2003

17 Heimola, Antti. ASTE Introduction. Company's internal document. ST_ASTE.ppt –presentation. Accessed 10 May 2009.

18 Lappalainen, Pertti. ATS3 Quick overview. Company's internal document. ATS3_Quick_overview.ppt –presentation. Accessed 11 May 2009.

19 Heimola, Antti. Test tools & Automation introduction. Company's internal document. ToolsIntro_June2006_0.1.ppt –presentation. Accessed 26 April 2009.