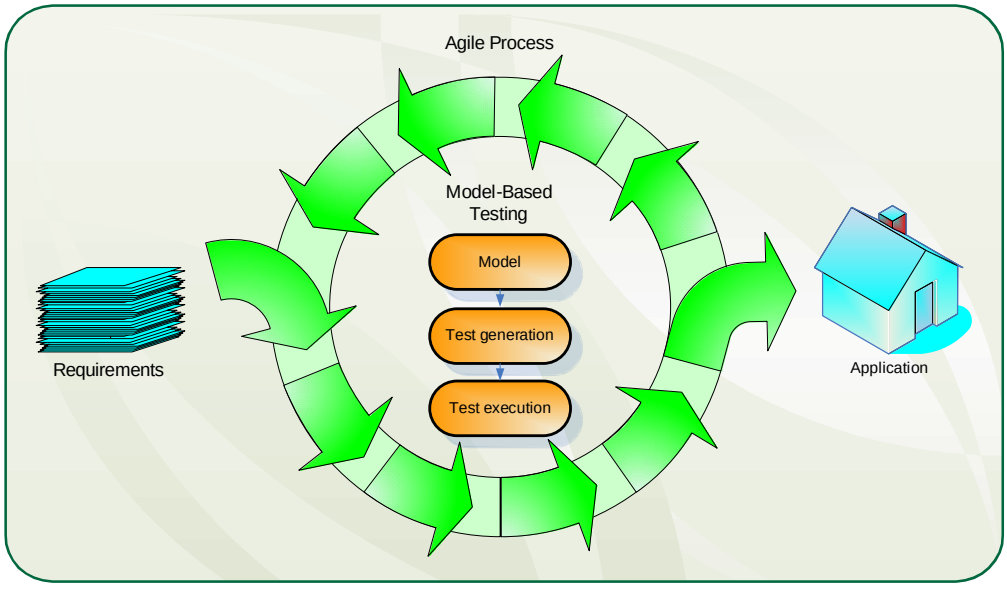
ESPOO 2008 VTT PUBLICATIONS 694



Olli-Pekka Puolitaival

Adapting model-based testing to agile context

VTT PUBLICATIONS 694

Adapting model-based testing to agile context

Olli-Pekka Puolitaival

ISBN 978-951-38-7119-2 (soft back ed.)
ISSN 1235-0621 (soft back ed.)

ISBN 978-951-38-7120-8 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 3, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax + 358 20 722 4374

VTT, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde 020 722 111, faksi 020 722 2320

VTT, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel 020 722 111, fax 020 722 2320

VTT Technical Research Centre of Finland, Kaitoväylä 1, P.O. Box 1100, FI-90571 OULU, Finland
phone internat. +358 20 722 111, fax +358 20 722 2320

Technical editing Leena Ukskoski

Edita Prima Oy, Helsinki 2008

Puolitaival, Olli-Pekka. Adapting model-based testing to agile context [Mallipohjaisen testauksen soveltaminen ketterässä ohjelmistokehityksessä]. Espoo 2008. VTT Publications 694. 69 p. + app. 6 p.

Keywords software testing, testing automation, software developing

Abstract

This study concentrates on model-based testing in agile software developing context. Model-based testing is a software testing technique in which tests are generated from a model. Test can be executed separately later or in motion during the generation. Special focus is placed on examining the adaptability of model-based testing to agile software developing context.

The purposes of this study were to find guidelines for model-based testing tool selection and to evaluate most suitable tool in agile context in case study. First was performed literature survey, where found criteria for model-based testing tools selection. Based on literature survey, was analyzed available tools carefully. Based on literature review and evaluation was made a collection of guidelines for tool selection and selected one tool for case study.

The case study aims to evaluate model-based testing suitability for agile developing project. This case study had two purposes: the first goal was to present model-based testing usage in agile process, and the second goal was to evaluate model-based testing suitability in agile context. Based on empirical findings, it was concluded that model-based testing can be performed in agile process.

Puolitaival, Olli-Pekka. Adapting model-based testing to agile context [Mallipohjaisen testauksen soveltaminen ketterässä ohjelmistokehityksessä]. Espoo 2008. VTT Publications 694. 69 s. + liitt. 6 s.

Avainsanat software testing, testing automation, software developing

Tiivistelmä

Tässä työssä käsitellään mallipohjaista testausta ketterässä ohjelmistokehitysympäristössä. Mallipohjaisella testauksella tarkoitetaan tekniikkaa, jossa mallista tuotetaan testejä. Testit voidaan ajaa myöhemmin erikseen tai testata ohjelmaa sitä mukaa, kun testejä generoidaan. Työssä keskitytään tutkimaan mallipohjaisen ohjelmoinnin soveltuvuutta ketterään ohjelmistokehitykseen.

Työn tarkoituksena oli sekä etsiä suuntaviivoja mallipohjaisen testaustyökalun valintaan että tehdä tapaustutkimus parhaaksi valitun työkalun käytöstä ketterässä projektissa. Ensiksi suoritettiin kirjallisuuskatsaus, jossa etsittiin kriteerejä mallipohjaisten testaustyökalujen valintaan. Kirjallisuuskatsauksen perusteella analysoitiin saatavilla olevat olennaisimmat mallipohjaiset työkalut huolellisesti. Analyysin ja kirjallisuuskatsauksen perusteella tehtiin kokoelma suuntaviivoja mallipohjaisen työkalun valinnan tueksi ja valittiin yksi työkalu tapaustutkimusta varten.

Tapaustutkimuksen tarkoitus oli arvioida mallipohjaisen testauksen soveltuvuutta ketterään ohjelmistokehitykseen. Arvioinnilla oli kaksi päämäärää: kuvata mallipohjaisen testauksen käyttöä käytännössä ketterässä projektissa sekä ar-vioida mallipohjaisen testauksen soveltuvuutta tähän ympäristöön. Tapaustutkimuksen perusteella ketterässä ohjelmistokehitysprosessissa voidaan tehdä mallipohjaista testausta.

Preface

This thesis was carried out as a part of the RITA (Rapid, Iterative, model driven Testing in Agile context) project at VTT Technical Research Centre of Finland in the group of Software Platforms 2007.

I would like to express my gratitude to all of those people helping and supporting me with this work. I would especially like to thanks Tienoo project members and MR. Teemu Kanstren for giving me the opportunity to conduct this thesis.

I would like to thank my supervisor at the University of Oulu, Professor Juha Röning and Janne Haverinen, for comments, suggestions, and having the time to review this work. Finally, I would like to acknowledge the important role my parents and sisters played as they encouraged and supported me with my studies and work.

Oulu 10.9.2008

Olli-Pekka Puolitaival

Contents

Acronyms and abbreviations

API	Application Programming Interface
CQ	Confomiq Qtronic, a model-based testing tool
DSM	Domain Specific Modelling
FACMA	Mobile Facility Management Services, a project in VTT
FSM	Finite State Machine
GPS	Global Position System
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
JML	Java Modelling Language
JSP	Java Server Pages
J2ME	Java 2 Platform, Micro Edition
JUnit	A unit testing framework for the Java programming language
JWebUnit	A Java-based framework for testing web applications
LTD	LEIRIOS Test Designer, a model-based testing tool
MaTeLo	Markov Chain Logic, a model-based testing tool
MBT	Model-based testing
Mobile-D™	Agile method made by VTT

NFC	Near Field Communication
OCL	Object Constraint Language
PIW	Post Iteration Workshop
QML	Qtronic Modelling Language
RFID	Radio Frequency Identification
RITA	Rapid Iterative model driven Testing in Agile context, strategic project of VTT
SUT	System Under Testing
Spec#	Spec Explorer using language for describing system features
SVN	Subversion
TDD	Test Driven Development
TTCN-3	Testing and Test Control Notation, version 3
UML	Unified Modelling Language
VTT	Technical Research Centre of Finland
XML	Extensible Mark-up Language
XP	Extreme Programming

1. Introduction

Already in the 1960's software developing processes were in crisis [1]. Projects were late, too expensive and the necessary quality was lacking. Attempts to solve those problems were made via several large software developing process models. Early on those proved to be too complex and maintenance-intensive. First of all the processes were too inflexible. In reality, the customer is continuously changing requirements and those changes cause problems and increase workloads. Nowadays these problems still exist and they are even worse than before. The importance of software is growing all the time and the problems related to software developing processes have become everyday phenomena. Agile methods constitute the latest solution for solving software process problems. While the Agile framework cannot solve all the problems, it provides a framework that improves the possibilities for successfully completing software projects. Agile methods have been taken into use with great success in industry. One pilot project has already reported a 70% cost saving. [2]

The development of software developing methods has posed challenges for software testing as part of software quality verification. There have been numerous attempts to improve software quality verification in several ways, which have been variously tested in history. The first way of performing testing was manual testing, which is still widely used. Manual testing proved, however, to be expensive, when the same tests were made several times. Capture and replay testing has been done to reduce the cost of test re-execution by capturing the manually run test and running it again. The maintenance of capture and replay testing was, however, costly because after every small change in the system's external behaviour the whole test had to be captured again from the beginning. Capture and replay tools are still used for testing some graphical programs but are no longer widely in use. The next attempt for raising test automation and reducing the effort around the test process was script-based testing. Scripts are able to run whenever required and this made it possible to perform full-automated execution. In practise, test scripts have grown almost as big as the whole program and the size produced maintenance problems. Data-driven testing, table-driven testing, action word testing and keyword-driven testing have tried to solve the maintenance problem by raising the abstraction level of test cases. Model-based testing is the latest solution to solve software-

testing problems. Model-based testing does not solve the entire problem but it aims to solve the automation of design functional test cases, reduce maintenance costs and automatically generate a traceability matrix. [3]

MBT has demonstrated various advantages and yielded good results in the research community and industrial case studies [4–8]. Despite the success of studies, model-based testing is not considered as widely accepted practice within industries. Thus, the adaptation of MBT has been slower than expected. Based on our experiments, the main reasons today could be a lack of knowledge of model-based tool selection and the finding of good practices.

Adoption processes for software testing automation are difficult and most of the time result in failure. MBT is not exception. The applied field of MBT for commercially available and applicable tools is fragmented, resulting in difficulties finding a suitable tool that fits in the testing domain and the system under testing (SUT). Another reason appears to be that several MBT evaluations are mostly performed by researchers using their own chosen tools within a narrow context in certain projects. However, there is lack of adequate guidelines for the selection and usage of MBT tools in different projects and processes. MBT algorithms and theory are well described in several places but there is no guideline for making them in practise. In this thesis the author aims to help MBT adoption for industrial use by providing guidelines for MBT tool selection, evaluation descriptions of most suitable MBT tools and presenting a case study for the adoption of MBT in an Agile context.

2. Agile software development

Agile software development methods are the latest attempt to solve changing requirements, exceeded time limits and quality deficiencies in software development. Agile methods have shown good results in a number of case studies and their adoption seems to be on the increase. This chapter describes the background to agile methods and Mobile-D™.

2.1 Agile overview

Agile software development methods provide fundamental practices and principles. Agile is an iterative process at the heart of which is a self-organizing team. Agile processes are not possible to copy straight from team to team. Agile is high-level framework and thus the team has to find out its own best practices to work in the agile way.

During the 1990's various so-called lightweight software development methods came into being. These lightweight methods have been the base for agile methods. The true "Agile Movement" in the software industry came into being in 2001 when 17 lightweight methods specialists gathered in Utah. They discussed the commonalities of their methods. They developed the agile concept and wrote an agile manifesto to describe the main elements. The manifesto was written as follows.

"Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others to do it. Through this work, we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more." [9]

“Individuals and interactions over processes and tools” signify that human roles are the most value thing. Hence, the process is adapted for people as well as possible. This can be seen for example, in the concept that overtime working is not well advised. Agile methods emphasize interaction and collaboration between team members. Discussion is a more comfortable and effective way to communicate for people than reading text and writing. Collaboration means really to do the same thing together. In this way more members know things collectively and the one person lacking in that knowledge is no barrier to developing ideas further. [10]

“Working software over comprehensive documentation,” means focusing on the software features realization. Documents are not worthless but the main purpose of the developer team is to produce working software. Team member knowledge is more important than the details in a document. This way they can focus on system development. Only the required documents are written. [10]

“Customer collaboration over contract negotiation” means iterative development and collaboration with a customer instead of making a plan before the project and following that to the end. In practice, the development process is divided into small iterations. After each iteration, the working application is demonstrated to the customer and the customers then say what will be implemented in the next iteration. When the customer sees the application it is easier to say what is wrong than plan everything before the project. During the iteration, the situation is stable, because changes can be made only between iterations. The duration of an iteration can be from one to six weeks. [10]

“Responding to change over following a plan” is has to do with Problems with changing requirements. The traditional software process is based on requirements that are given at the beginning of process. Everything is based on these requirements and changes that are made are costly. In practice, these requirements are changing all the time. Agile software developing methods are based on the logic that everything is changing during the process. The agile process is like doing small projects, which are described as iterations. Everything can be changed between iterations. In fact, changes are seen as good thing. The customer does not have good vision of product at the beginning. During the agile process, the customer is given every possibility to define his

needs with greater clarity. In this way it is possible to develop more favourable client relations. [10]

2.2 Testing in agile process

Agile processes mostly define positive unit testing. Agile methods usually bind the testing as part of the development tasks. The developer writes the code and a couple of positive test cases, which test the ideal use case. The tests have to be written before the task can be finished. Written tests are set as part of a regression test suite. The regression suite has to be successfully run, before the code is committed for common version control. This way, developers can be more assured that the new code is not broken.

Test-driven development (TDD) means writing tests before the code [10]. When an engineer employs TDD, he/she writes one or more positive tests before implementing the code. Test writing is like planning the code part of external behaviour. It is natural to do planning first and then continue with implementation. TDD ensures that tests are really written and the regression test suite exists. TDD is an original XP method, but it has been used widely.

2.3 Mobile-D™

Mobile-D™ is a process model for agile development [11]. Mobile-D™ is not exactly determined. Therefore, every team conducts it slightly differently In this chapter our style of performing Mobile-D™ is presented and developed into a case study (see chapter 5).

Mobile-D™ is an agile method, which was empirically composed over a series of software development projects in 2003–2006 [12]. The method is based on a two-month production rhythm, which is divided into five sub phases. Each of the sub phases takes from one to two weeks. These phases are called setup, core functionality one, core functionality two, stabilize and wrap-up & release. Mobile-D™ adopts most of the Extreme Programming practices, Scrum management practices and Rational Unified Process phases for lifecycle coverage. The method is described in pattern format and can be downloaded from [13].

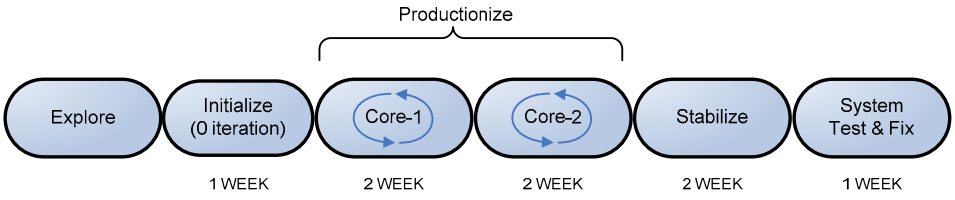


Figure 1. Mobile-D™ working process [13].

2.3.1 Overview

The Mobile-D™ working phases are presented in Figure 1 and described below [13]:

- **Explore** phase practically means establishing project environmental dependencies. That includes establishment of stakeholders, scope defining and project establishment. Stakeholders are customer interest groups and other people who are interested in the project. When the stakeholders group is established it is possible to create initial versions of a requirement collection and a project plan. Those determine the high-level purpose of the project. Then the environment can be selected, personnel allocated, the architecture line defined and the process established.
- **Initialize phase** or zero iteration means establishing project set-up. In this phase, the developing team starts its work. They build up a developing environment and make architecture planning. At the end of this phase everything is ready for the first developing iteration. The most important things are set-up, the physical and technical resources for the project as well as the environment for project monitoring, training the project team up to the needed level and establishing project specific ways to communicate with customers. Mobile-D™ is a general framework and does not give a description for environment set-up or team training. It is important to establish customer communication so that both the customer and the developer team get the necessary information fast enough, in an appropriate manner and in a useful format.
- **Core-1 and Core-2** develop iterations. The core phase's purpose is to implement the required functionality from sprint backlog to the final product. Iteration is two weeks long and consists of one planning day,

eight working days and one release day. Implementation iteration phases are described more specifically in chapter 3.3.2.

- **Stabilize phase is an iteration, which aims to integrate** subsystems into a single product. Big software normally consists of small parts. These parts are collected together in stabilize phase.
- **System test & fix** is the last phase of Mobile-D™. The purpose of this phase is to make more tests, fix faults and produce documentation. The product quality has to be adequate and software without documentation is unusable. Source code is not enough for communicating the software's features, structures and so on. Documents will be produced for project stakeholders and not for the agile team.

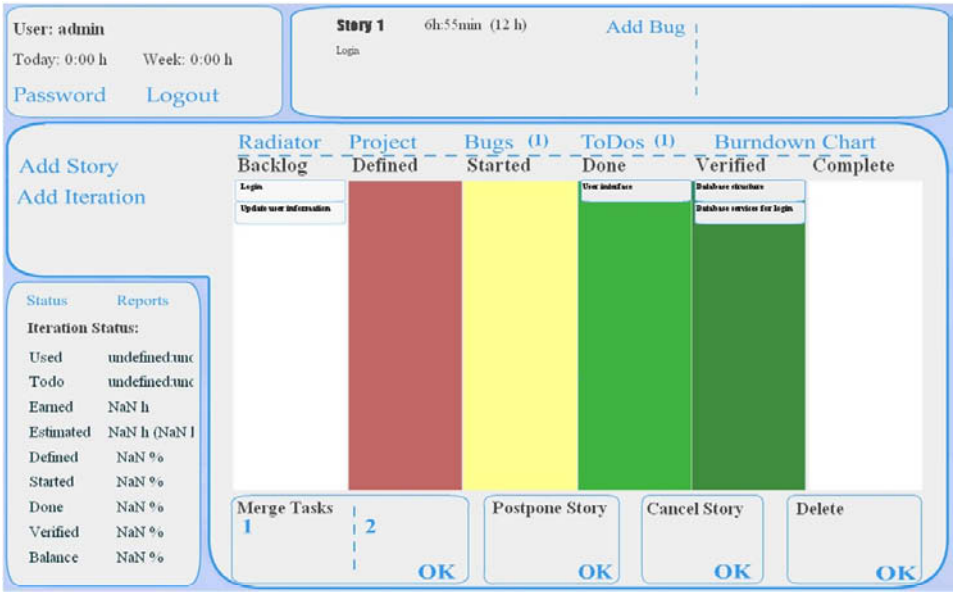


Figure 2. Task Master.

All agile methods handle tasks almost the same way at a high level. Every method has its own practises for task handling at a lower level. Mobile-D™ uses Taskmaster. Taskmaster is a tool for task position management and presents the information of the iteration's current condition. That information is highly useful for team members and interest group members. Taskmaster is described in

Figure 2 [13]. Taskmaster includes five columns the meanings of which are described below.

- **Backlog** state is for stories that are selected for implementation in the current iteration. Stories typically mean program features. Stories are divided into tasks. The last four states visually describe the tasks state in developing.
- **Defined state** is the starting state for tasks. When an engineer begins a task he or she is moving that from the defined phase to the started column. This way everyone else knows that someone is implementing that task. The advantages are that people are not doing the same thing and know better the situation of the iteration.
- **Done state** means that the code is written, but also that positive unit tests have been successfully run.
- **Verified state** means that the test engineer has verified the task. The test engineer checks that all the tests have been made and completely run.

2.3.2 Implementation process

The Roles inside an agile developing team are considered different from traditional software developing methods. Traditionally, team members have had specific roles. Traditionally thinking suggests lead team members specialization. Testers only test the program and developers implement code. This thinking has led to a situation where developers have specialized in some part of the code and they alone have been able to develop that part. When the person in question changes company or has been unable to work, that part of the program's developing has stopped. The purpose of agile methods is to share knowledge through the team. This approach makes it possible for everyone in the team to develop every part of the code. Consequently everyone has wider skills and knowledge and development does not stop when one person from the team is absent. Members also have a better picture of the whole and can help each other work with several decision processes at a time. [13]

Mobile-D™ has no roles but it does have responsibilities. Roles are more binding to persons than responsibilities. With a role, the person is doing things

that belong to his/her role. With responsibility, the person must take care that all things are performed that belong to his/her area of responsibility, and the person must do those him/herself. Responsibility is similar to the supervisor's role. Mobile-D™ traditionally uses developing role names for naming responsibility areas, because these are easier to adapt. Mobile-D™ responsibilities are for example: Team Leader, Tester, Metrics, Architecture, and Quality Engineer. Mobile-D™ directs the team towards self-organizing. Therefore, responsibilities are not needed before the project, but those can be set continuously during the project when some responsibility is needed. Self-organizing means also actively searching out better practices and developing the process.

The Mobile-D™ core phase's iterations consist of three parts these being: planning day, working days and release day. These parts are described in Appendix 2 and more below [13]:

- **Planning day:** All iterations start from a planning day. During the planning day the purpose is to select and plan the work contents for the next iteration. The customer participates actively and ensures that the requirements provide the most business value, are identified and that those requirements are correctly understood. At the beginning of the day the requirements are analysed. The purpose of the requirement analysis is to analyze, determine priorities and select carefully requirements for each iteration in collaboration with the customer. After this, we have a prioritized product backlog for at least the next iteration. The product backlog is used for iteration planning to share features in stories and stories to tasks and evaluate those efforts. Te evaluated implementation time is written in the task card. Later the real implementation time will be written also, so it is easy to judge the accuracy of the evaluation.
- **Working day:** The working day is normally started as a wrap-up. A wrap-up is briefly an informal daily meeting for increasing awareness and process control. The purpose of the daily wrap-up is to share knowledge among the developing team. Everyone says what he or she requires other people to know. The team can find their own best way to keep wrap-ups. Scrum daily meeting questions can be used for helping members to notice wrap-up idea. Some typical scrum daily meeting questions are presented below.

1. What I have done since the last meeting?

2. What am I going to do before the next meeting?

3. What impediments do I have?

The wrap-up can be kept daily or every second morning. The e time taken and its intensity is the team's own decision. Mobile-D™ strongly recommends pair programming, continuous integration and refactoring during implementation. Pair programming means that two persons are coding the same code at the same time. The purpose of pair programming is to improve communication, enhance process fidelity and spread knowledge within the team, and ensure quality of the code. Continuous integration of new code with the existing code in the code repository facilitates control over the constant change of software. Refactoring is improving existing code without modifying its external behaviour. Refactoring makes software more modifiable, extendable and readable. In the agile process all attempts are made to keep the process transparent. Transparency means team internal communication as well as providing an honest view of progress to the customer. When the customer is conscious of the process situation, he/she can give feedback on implemented features and guide development.

- **Release day:** Release day starts with testing and then continues with release ceremonies and finishes with a post iteration workshop. On the morning of the release day, the system is integrated and built-up, so it is a good time to check the quality in testing. Release day tests normally consist of acceptance testing, because the main goal is to ensure that customer-specified features are implemented correctly. The release ceremonies aim to present features for the customer. The customer tests the product him/herself and gives feedback. All realized bugs are written into the bug list and are fixed at the beginning of the next iteration. Development proposals are logged and the customer can use those in the next iteration backend.

3. Model-based testing

Model-based testing term is used for a wide variety of test generation techniques. In this thesis, MBT is defined as follows. Model-based testing is one particular type of software testing technique in which test cases are generated automatically from a model, which describe the behaviour of the system under testing from the perspective of testing. The model normally consists of states, triggers and expected outputs. The model describes the system's requirements for a test generator. The test generator generates a test suite automatically from the model. Tests fail when the expected behaviour based on the model is not equal with SUT behaviours. [3]

The main advantages of using MBT are: 1) Less faults in test cases because test generation is automated via sophisticated algorithms and there are no human faults in test cases. Human faults can still be found in the model, but those are easier to see. 2) The quality of the test suite for complex systems is better, because generation is made via sophisticated algorithms and there is no human limitation for designing different tests. 3) A number of faults will be found earlier, already in the modelling phase. The test model is at a higher abstraction level than the design model and therefore faults are made visible. 4) Non deterministic system testing and infinite test suites are possible with the online MBT approach. [3]

MBT has had slow adoption for industrial use. One reason is that MBT differs so much from other testing styles. MBT is software testing system development and neither test case writing nor execution as testing is seen normally. Therefore an MBT engineer must be a technical person. Normally, testers are non-technical and are unable to perform MBT successfully. Often testing is required to be immediately ready for use and MBT needs some preparation before using it. Additionally, the software industry has had a lack of MBT knowledge and metrics are not directly compatible with MBT. [14]

3.1 Online vs. offline MBT approach

The MBT can be divided into two different categories; online and offline testing. Offline testing signifies test suite generating from the model and its later execution. The export format of generated test cases depends on the used execution tool, and can be, for example a test script. In the online test generation approach, tests are generated and executed in motion. With online testing, it is possible to react to continual changes, and make autonomous decisions. This makes it possible to test non-deterministic systems and run infinite test suites [15, 16]. A comparison of online MBT with offline MBT is presented in Figure 3 [3, 17]

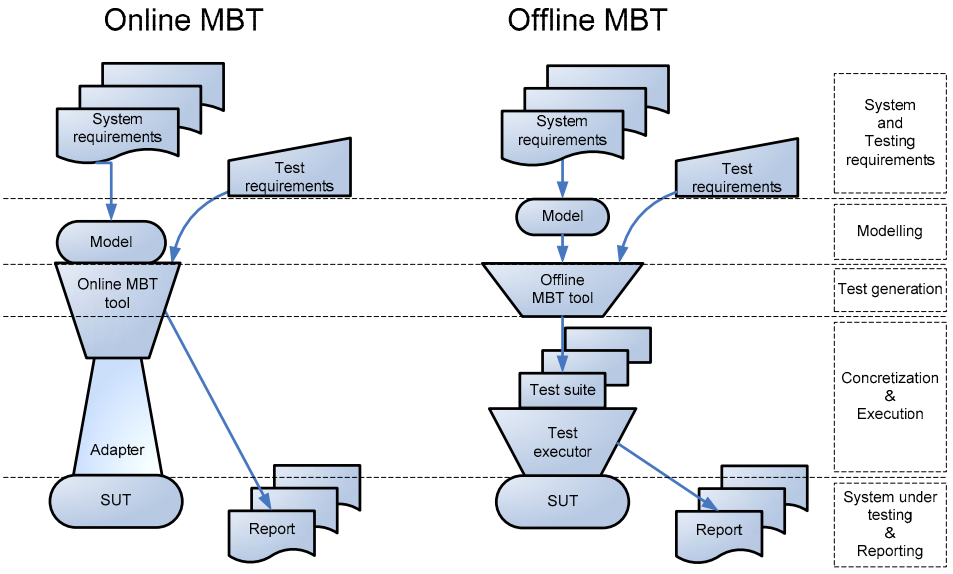


Figure 3. Online vs. offline MBT approach.

3.1.1 Offline model-based testing

Offline MBT means automated test suite generation from a given model and tests execution separately. Offline MBT testing process is described in Figure 4. The target system's behaviour is described in an informal requirements document. A model for test generation is made from the requirement specification. The model is imported to the test generator. The test generator generates test suites from the model with test requirements. Test requirements

guide the generator to make the required kind of test cases. Requirements are for example, coverage criteria or targets in the model. The generated test suite is entered to a test executor. The test executor runs test cases against the SUT and makes a report from the results. The executor is usually an external tool. This chapter is based on [18].

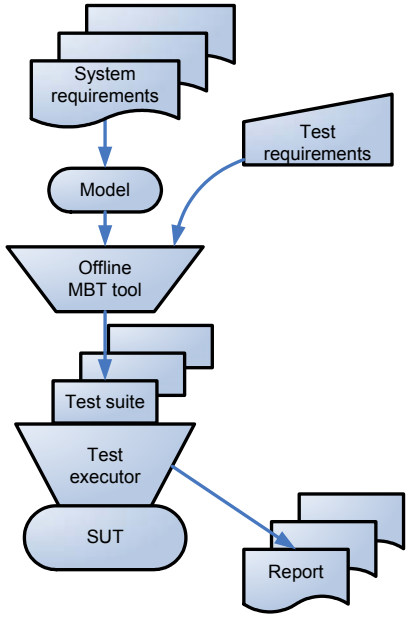


Figure 4. Offline model-based testing process.

The main advantages of the Offline MBT approach are 1) automatically generated test cases, 2) easier adaptation for program changes, 3) good adaptability for existing tool chain and 4) adaptation layer reuse.

Automatically generated test cases save efforts and increase the quality of the test suites. Model-based test generation is done using several test generation algorithms and strategies. These algorithms and strategies generate better test suite quality for complex systems than human can do. When the machine makes tests there are no human errors. Humans can introduce faults to the model, but the model is set at a high abstraction level and so it is easier to see faults.

Offline MBT test suites can be stored and run anytime without regenerating the test suite. Therefore, it is possible to use the generated test suite for regression

testing. When the program changes one only needs to change the model and regenerate a test suite.

Offline MBT is able to be adapted as part of a tool chain. Typically, program-developing companies are familiar with a modelling tool because they use them for developing. Mature MBT tools allow for the import of third party models and thus it is possible to use an already familiar modelling tool for MBT. Often, companies make a test execution platform for manually written test cases. Offline MBT tools can generate tests in different formats. Therefore, the generated test can be used on the existing test execution platform. These features make it possible to adapt offline MBT in an existing tool chain. Online MBT cannot normally be adapted for an existing test execution platform. The reason for this is that it is more difficult because the test must be executed in motion and MBT could receive the output value.

An Offline MBT generator generates abstract test cases, which have to be made executable before running. Making such executable is normally performed so that the generation tool writes tests in a format acceptable to the execution tool and the test execution tool then runs tests against SUT. Therefore tests are made executable partly in generator and partly in executor. The main thing is that performed test executions can be fully reused in the same test execution platform.

3.1.2 Online model-based testing

Online MBT means easily automatic test suite running against SUT, so that the next test step is generated when the previous test is executed and a response value has received. Online MBT is seen as a game between the MBT tool and the SUT. The Online MBT process is described in Figure 5. [17]

The Online MBT process starts in the same way as offline. The high abstraction model is created based on program requirements. Then the model and test requirements are imported to the MBT tool. In online MBT, a test generator and an executor are found in the same tool, because of the possibility to make tests generation and execution in motion.

Test requirements include lighter algorithms than the offline approach. In the online approach, the test requirements mean normally walking algorithms. Walking algorithms used for test generation are useful when a long time test session is performed. Before online MBT can be started, the adaptation layer has to be implemented. The online adaptation layer joins the SUT and MBT tester together. When the model for testing is imported, test requirements are defined and the adaptation layer is implemented, it is possible to start the test. The online MBT tool is tested continuously in motion, which means forwarding one-step in the model, running that step immediately in the SUT and analysing the results. If the value differs from what it is expected, based on the model, the test fails.

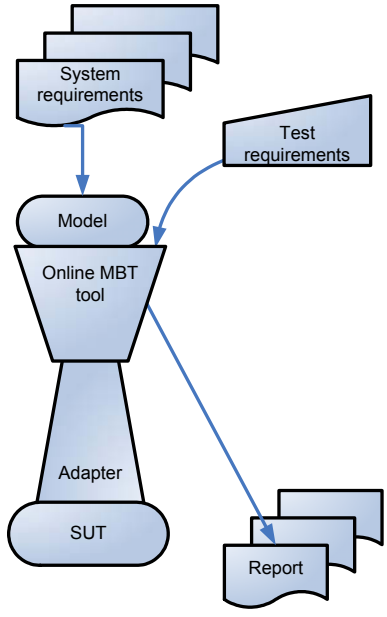


Figure 5. Online model-based testing process.

Compared to the offline approach, the main advantages of online MBT are running infinite test suites and testing non deterministic systems. The online model-based testing approach is connected directly and continuously to the model and this makes it possible to react continuously to changes and perform autonomous decision-making. Therefore, testing of non-deterministic systems is possible. By using online testing, it is possible also to make the testing session as long as required, or until the program crashes. This is especially useful when there is a need to test for example, memory leaks over a long period. [18]

The online tool can also write log and that can be used as a document. The log can be used also later to run the same test suite again.

Compared to online testing the biggest difference with offline testing is the delay between test generation and execution. Both of those automatically provide test generation and validation based on an abstract model. That means saving efforts in test writing, test suite maintenance and test suite adaptability for changes. The offline approach is easier to adapt in existing software developing because of this independent test suite generation and execution. This makes it possible to use the existing test execution platform. Online testing has a bigger gap for the normal testing process because of simultaneous test designing and execution. Therefore, the adaptability for existing tool chains is more problematic and this may be a reason why the online approach has been less adopted for industrial use. [3]

3.2 Modelling

In MBT, the model presents system requirements for the test generator. The system requirements are normally presented in an informal document. The test engineer translates the requirement document wholly or partly in test model format. The model is the only knowledge of the system that the test generator has. Modelling strongly affects the required amount of effort and quality of tests.

A number of cases prove that the modelling phase determines almost as much faults as test running. This is possible because test requirements form an informal document and from this it is harder to see crosswise requirements, which are not possible at the same time. When the system is modelled at a high level, these conflicts become visible and can be fixed. Humans can also see faults more easily from a high-level model than from requirement documentation. [3, pp. 59–106]

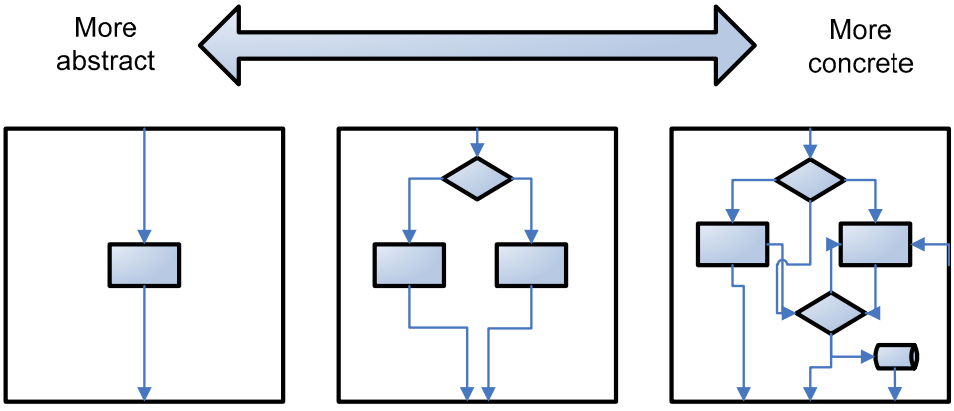


Figure 6. Abstraction level.

The first step in modelling is to choose what to model and from which aspect. Modelling can be started from small part of the SUT and then built up step by step. There is no need to model everything before starting the test. It is also important to select a good level of abstraction. That means deciding how many details are included in the model. The abstraction stage is described in Figure 6. The model is used only for testing. To be efficient, the model must be much simpler than the full implementation model. Often, it is more practical to make several smaller models for individual components. Then the system level testing model can be simpler. To be useful for test automation, the model has to include excepted output data. Because of the inclusion of expected output data to the model, it is possible to compare output and expected output. That enables automatic faults detection, also known as oracle. [19]

The model for MBT describes SUT behaviour. Modelling can be made from two different aspects: test model or design model aspect. A test model aspect is a description of the SUT environment and a design model aspect is a description of the SUT itself. In a server client system when the server is the SUT, the test model is a description of the client and the design model is a description of the server as is described in Figure 7. The test model can be seen as a mirror image of the design model. Therefore, the test model's input value is the same as the design model's output value. These are equal approaches from the test generation viewpoint. Design model aspect usage provides possibilities for reuse models, which are made for the software developing purpose. [19; 3, pp. 59–106]

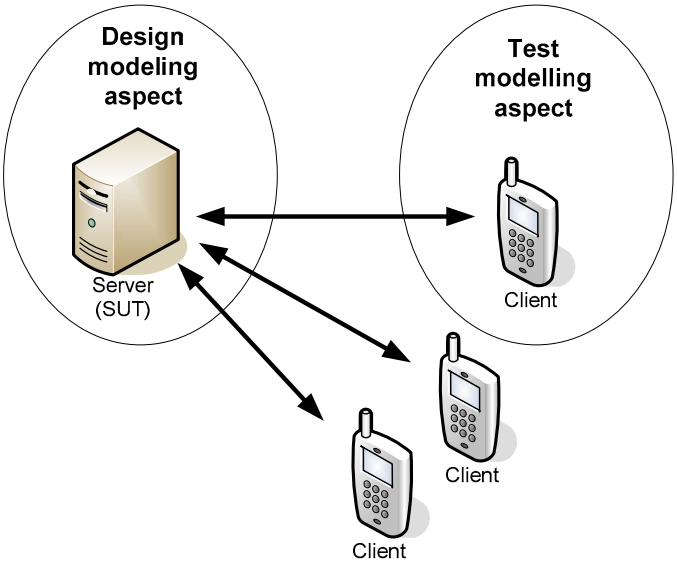


Figure 7. Modelling aspects for MBT.

Modelling notation has a strong influence on modelling efforts. The model describes system behaviour. Therefore, if the notation is more effective for modelling the domain of the SUT, it saves efforts. The model notation's domain means the area and which kinds of systems can be modelled with that. Generally, narrower domain notations are more efficient. Very narrow notations are called domain specific modelling (DSM). Actually, DSM means making a new notation for every single system. DSM notations are most efficient for modelling because those are performed for describing the current domain's problem, rather than describing all possible domains. DSM is used in MBT for embedded software. Usually, generic notations are not so effective in any single case but enable the modelling of almost everything. At present, general modelling notations are most used in MBT. [17]

Several notations that are currently used in MBT (i.e. UML) are made originally for software system design rather than for software testing. Therefore, these modelling languages are not native for testing. Test-specific languages such as TTCN-3 are better for modelling efficiently and testing purposes. Implementation oriented modelling languages have been the natural selection because of the test engineering familiarity stage. This can be seen also as a reason for increasing usage of design models in MBT tools. [17]

At present in MBT, most used model notations types are pre/post models and transition based models. Pre/post models consist of operations that have precondition and post condition. These are used design modelling most popularly. Examples of such notation include UML+OCL, B notation, Spec# and java modelling language (JML). Pre/post models are best for data-oriented modelling. Transition-based notations are models that usually include nodes and transitions like finite state machine (FSM). Examples of these are UML state flow, Simulink Stateflow charts and Markov Chain models. These are best for modelling control-oriented systems. These kinds of models have problems for including data knowledge. Sometimes these are extended with some programming language for presenting data. There are several more classes of modelling notations. These two are most essential to this work and therefore others are not presented here. Further reading on this subject can be found from [3, pp. 59–106].

Model validation is important because there may be human faults. Usually, the MBT tester checks the format correctness but also checks logical impossibilities. Logical impossibilities can be for example: unreachable states or a transition that does not have a target. From a complex model it is hard to see if it does not fit the system requirement correctly. Various MBT tools are available for providing test cases visualization in sequence diagram format, and for facilitating manual validation. From a sequence diagram is easy to see the correctness of generated test cases. If test cases can be demonstrated to be impossible, the model is not correct. When the model takes the form of a concrete design model, it is possible to animate it. 'Animate' here means for example, running the model systematically. Then the variables values tell if it works correctly. Validation methods help model error detection before the test suite is generated, and reduce faults in test cases. [15]

3.3 Test generation

The main benefit of MBT is automatic test generation. Automatic test generation requires a description of the system and test requirements. A description of the SUT is the model and test requirements guide the test generation. Via sophisticated test requirements, it is possible make more efficient test suites and the generation can be faster. The collections of test generation guiding algorithms vary a lot in

tools. A great number of algorithms is, however, not always beneficial. This chapter is concerned with how to facilitate MBT tool selection for describing the test requirements. Test requirements can be divided into three main categories: coverage criteria, targets and walking techniques. [3, pp. 110–138]

Coverage criteria show how well a generated test suite fulfils the model. Coverage criteria are mostly used in offline MBT, because it is not so time critical compared with online MBT. Some tools also use coverage guidance for online MBT.

Coverage criteria used for test generation guiding mean that the test generator tries to cover given criteria as well as possible. Most coverage criteria algorithms and terms are adopted from white box testing where those mean code covering. In MBT, the coverage criteria mean how well a generated test suite covers the model. Therefore, in MBT, 100 percent coverage does not mean that code is 100 percent tested. 100 percent coverage means only that there is a generated test suite, the tests of which cover 100 percent of the model from a selected criteria aspect. [3, pp. 110–138]

There are several different kinds of coverage criteria. Presented below are five examples but more can be found:

- **State coverage** criteria measure how well a generated test suite covers model states. 100 percent state coverage means that at each state of the model is tested at least in one test of the test suite. This is one of the most used coverage criteria.
- **Transition coverage** means the model transitions covered in the test suite. Using this method it is useful to know how the tool handles two-way transitions. Two-way transitions can be required to test both directions or just one direction to be covered.
- **Boundary value analysis** makes tests that are near by value boundaries. Typically, every boundary is tested with three tests: one below, one equally and one over value. This can find errors when value boundaries are not well implemented.
- **Branch coverage** means covering model branches. All branches are covered when a test suite has at least one test suite for each branch.

- **Most probable route** covers the route, the probability of which is highest. This requires probability numbers in the model.

Targets are points in the model. A target is fulfilled, when a test is generated which passes the target. This is mostly used for offline MBT because it needs a lot of calculation power. The targets are good for making a small set of test cases or one particular test case. Targets are described also as requirements, because a target can be used for determining requirements in test cases. Targets placed in the model give good traceability. Several tools use a traceability matrix from which it is easy to see that which targets are tested in which test case. [3, pp. 110–138]

Walking techniques determine how the test generator walks through the model during the test generation. These are not normally used for offline testing because other guidance algorithms provide a better quality of test cases. These can be used in offline testing as a couple of other tools are doing. Added to this, MBT walking techniques are widely used in online because they are lightweight. With online MBT it is possible to perform non-deterministic testing and infinite test suites. In those cases is impossible to build searches because of its infinite size. Therefore, online MBT testing can be guided well with walking techniques, especially in very long test suites. Good examples are random walking, coverage guided and with probability numbers. Random walking is just randomly walking through the model. Coverage guided means trying to fulfil coverage criteria better than random walking. That means small searches maintained during the testing process and promoting coverage criteria fulfilling. Probability numbers can be used at least in two ways. First, one is to set static probability numbers for transitions and walking so that a higher probability number means taking that step. The second one is to use variable probability numbers so that every step taken decreases the current transition probability. This leads in the long run to wider coverage while the same route has a lower priority as the new one. [20; 3, pp. 110–138]

Time limits do not guide the test generation. Time limits are important for testing time-limited features. In online testing, time limits often means a response waiting time or test generation time. In offline MBT, time limits are included in exported test script. Then the test executor knows the time limits from the script.

Presented next is the most important thing about test generation in MBT based on our investigations. Tools provide much more features, but we determined those to be less important. Extra features can be found from tools user manuals. In summary, better test generation guiding makes more effective and test suites better fitted for their purpose. More is not always better with test requirements. The necessary test generation requirements are case-specific.

3.4 Making tests executable

When a test suite is automatically generated, is almost useful to increase the testing automation stage as automating tests execution. Generated tests are set at high abstraction state as the model for testing because the model is the only information from the SUT that the test generator has. Tests have to concretized before they cam be run against the SUT. Concretization is performed in the adaptation layer. [3, pp. 283–305]

The adaptation layer maps the MBT tool for the system under testing. The layer can consist of a single component, or a chain of components. Three different ways for implementing the adaptation layer are described in Figure 8.

The offline MBT approach generates an abstract test suite and does not directly take care of test execution. Case B is the normally used high-level structure of offline testing adaptation layer. In such a case high-level test script is exported and that is separately executed on the test execution platform. In case C, the exported test script is executable. That normally means that the MBT tested has some adapter inside, which includes the concretization knowledge. This is a possible solution with domain-specific tools. These are only three high level examples and in practice, there are countless variations of the adaptation layer structure. [3, pp. 283–305]

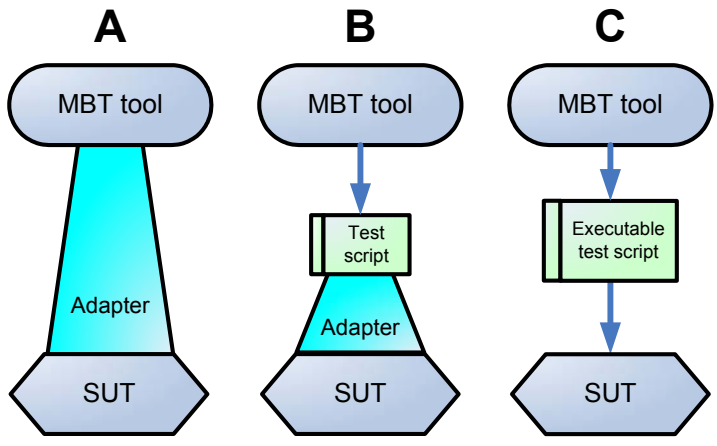


Figure 8. Adaptation layer [3].

The online MBT approach needs a continual two-way connection between the MBT tool and SUT. Then the adaptation layer could be a single adapter as in case A in Figure 8. The adapter may consist of several parts but it must be able to work faster than the offline adaptation layer. [21; 22; 3, pp. 283–305]

3.5 Test execution and reporting

System developers implement one interpretation of the system requirements. The MBT test engineer models the other interpretation. Test running tests whether these two interpretations match.

Usually, the offline MBT technique does not consider test execution. It exports the generated test suite in a determined format. It is possible to include a test executor in offline MBT tool but we could not find any example. Exported test suite execution styles vary from manual to fully automatic. Because the offline MBT tool does not execute tests, it does not have information of test success. Therefore, the offline MBT tool is unable to write reports. An MBT tool can write some reports describing the content of a test suite but test execution reporting is the test execution platform's task.

Online MBT also runs tests. Test execution takes place through the adapter. Thus, the adapter takes care that test information is transmitted to the right input

interface of SUT and the output data from the SUT is received. The online MBT tester is also suitable for test result analysis. Therefore, it can write test reports and export them.

3.6 Reusability

Reusability is always an interesting issue in software area because it can save so much effort. Reusability in MBT can be divided into three main aspects: implementation purpose model reusing in MBT, MBT model reusing and adapter layer reusing.

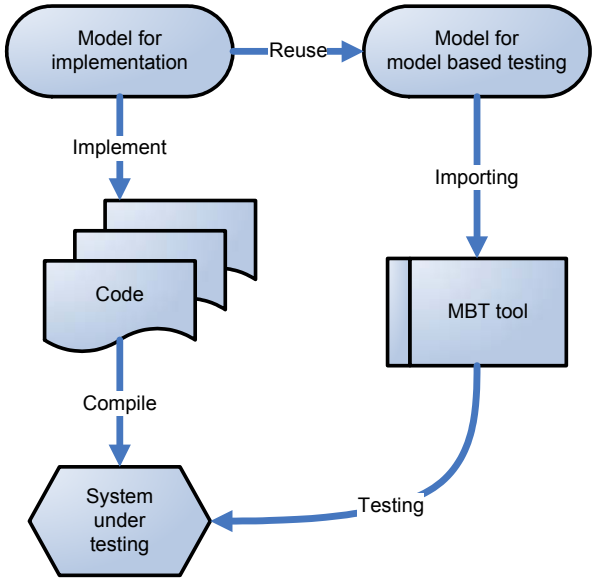


Figure 9. Implementation purpose model reusing for MBT.

Implementation purpose model reusing as a model for testing is presented in Figure 9. That is possible only with MBT tools which use an implementation-modelling approach for testing. This reuse aspect would save much effort because there is no need to make a separate model for testing. There is a risk that if the original model has a fault, so the fault is consequently included also in the test suite. This leads to a situation where the test suite can be successful run although the system has a fault. The further the models are from each other the

more probable it is that they do not include the same faults. The implementation model is a concrete description of the system, but a very high abstraction model is normally used for testing. That may lead to some problems with MBT tools and system logic understanding. [17]

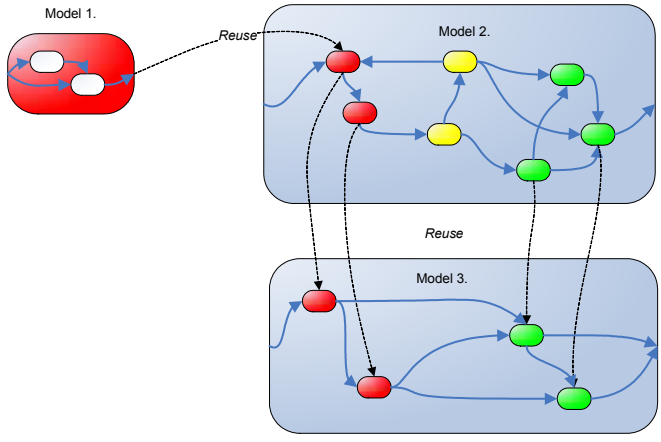


Figure 10. Models reusing in testing.

The model for testing reusing for another model is the same kind of reusing found in code line copying. The two main ways are models linkage and model partly reusing. Use of these depends normally much on features provided by the MBT tool. Model linkage means building a big model from small ones via linking. This approach is presented in Figure 10. There the first model is linked as part of the second model. Using this reusability approach it is possible to test smaller parts individually and collect those parts for testing wholeness. Model partly reusing is able to be performed in the same manner as copying in programming. That requires that the model be divided into smaller parts. The idea of model partly reusable is described in Figure 10 between the second and third model.

Adapter reusing is one of the most reusable things in MBT. The adapter layer translates an abstract test suite from the MBT tool executable in specific test execution platform. The MBT tool is relatively the same in the same developing or testing team and the test execution platform number is normally small. When the MBT tool and test execution platform are the same as the adaptation layer, it is fully reusable. That requires that the adapter is well implemented and fully able to cover the formats.

4. Model-based testing in agile context

This thesis aims to adapt MBT in the agile context. In this chapter MBT adaptability to the agile process based on theories is discussed. This chapter also gives hypotheses for the case study of this thesis. The results of the case study are described in chapter 6.

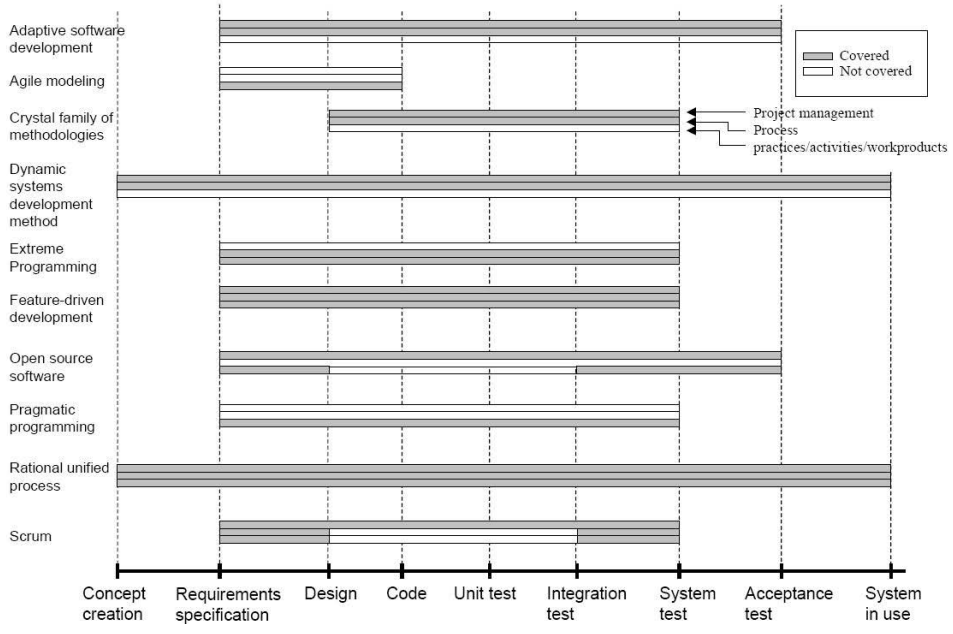


Figure 11. Software development life-cycle support [10].

Agile methods recommend unit testing in iteration and more tests afterwards [10]. Agile methods normally focus on software developing and thus lack testing practices. As Figure 11 presents, most agile methods do not determine acceptance testing. Most methods cover the system test. How are system tests covered? Extreme programming (XP) and Scrum are the most popular agile methods and those present the strongest evidence of their functioning [23]. Thus, these provide good examples of testing in the agile process. XP only recommends the developer to use TDD and the customer to write functional tests. The tester's role is to help the customer in writing functional tests. Scrum recommends unit tests made by the developer and system testing after the iteration. These practices are described more closely in chapter 2.2.

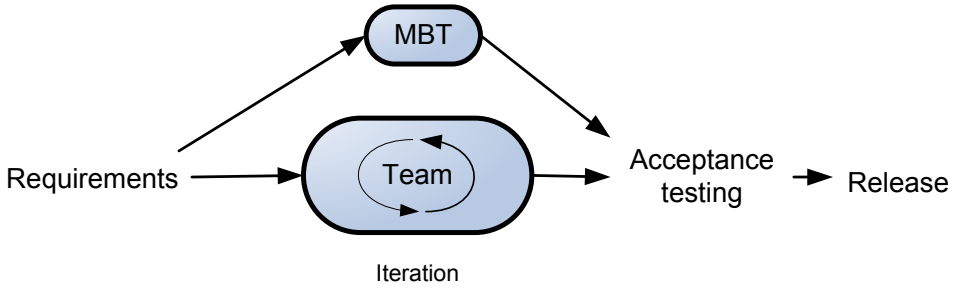


Figure 12. MBT outside from an agile team [3].

BT is thought to replace other testing styles and, to use as those other styles have used before. We could find only one source of MBT in agile [3, pp. 381–382]. In that source MBT suitability for agile process is discussed. The source related that MBT could be performed with XP in the same way that XP recommends for system level testing. It was also stated that the customer could possibly make the MBT model with the test engineer. The idea is to do MBT separately from agile developing team. This process is described in Figure 12. They also say that MBT can be performed in TDD way because it is usable for performing unit tests.

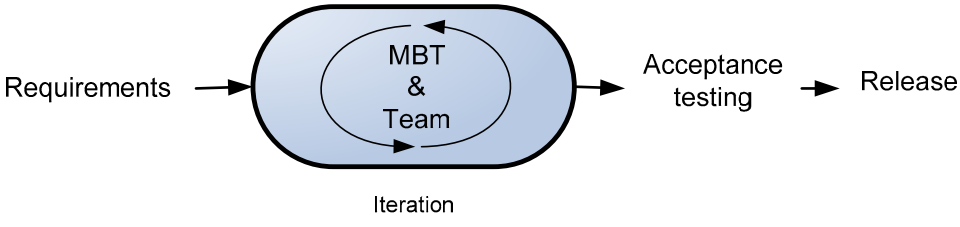


Figure 13. MBT inside an agile team.

We presupposed that MBT including an agile team is the best way to perform MBT in an agile context. In this thesis, the aim is to adapt MBT in an agile context as well as possible. The test engineer included in an agile developing team has improved software quality [24]. In this thesis, we aim to put the MBT test engineer inside the agile team as Figure 13 presents.

We presupposed that our approach would yield several advantages. In an agile project, things change a lot. If the test team is working separately, test engineers do not know about the latest changes. In agile developing, only the absolutely

necessary documents are made, thus the tester lacks overall documentation. This information gap can cause problems in the testing process. MBT putting inside the agile team should reduce these information problems. Agile development has good practices for software development. These should be able to be used also for testing system developing.

There are some challenges that can block MBT out of agile iteration. There is no existing answer for the question of what kind of testing can be performed in agile team We think that the most important thing is testing a style needing effort. If the testing is too heavy it should be left out of the team, because heavy things are not agile and these decrease team flexibility. The testing style has to be so light that it can be fitted in iteration. After every iteration, the application should be implemented and tested. If the testing needs too much effort, it is dropped out of the iteration. Then the software cannot be tested using that testing style before the iteration ends.

MBT testing is test system developing and not test cases writing. Agile methods have a number of practices for developing. Could it be possible to handle MBT in the same manner as developing? In agile methods, features are divided into tasks and the amount of tasks needing effort is estimated. It could be useful to put testing as a task and handle it in the same manner as developing tasks are handled. It could also be successful to perform MBT as pair programming and TDD.

5. Model-based testing tool selection

This thesis aims to facilitate MBT adoption for giving knowledge of MBT tools selection. In this chapter the aim is to clarify MBT tool selection for giving several MBT tools evaluation and guidelines for the analysis of MBT tools. Guidelines are based on model-based testing theory, which is described in chapter 3.

5.1 Overview of existing tools

In this section, we discuss a set of MBT tools from the viewpoint of the evaluation presented in the previous sections. The tools discussed have been chosen based on their maturity and commercial viability. A tool should have support available for its users, a well-defined user interface, and provide support for testing different kinds of applications.

For evaluation, we used evaluation licenses from the tool providers. Thus, this was a limiting factor as not all tool vendors were willing to provide an evaluation version of their tools. For most of the tools, these evaluation licenses could be acquired directly from the company web site or by request from a company representative. For some it was not possible and we had to base our study on other sources of information or leave the tools out. Thus, for these tools the analysis is not extensive and cannot be as objective as for the tools with the evaluation license.

For the Leirios Test Generator, we could not get an evaluation license as they considered their tool too difficult to learn and use without consultant training. As we still required the inclusion of information on this tool, we used the tool manual that was given on request and the information acquired from a video conference given by Leirios. T-VEC is another tool vendor that works in this area but they were not willing to provide any information or evaluation license for their tool. [25].

5.2 LEIRIOS Test Designer

LEIRIOS Test Designer (LTD) is commercial offline MBT tool provided by LEIRIOS [26]. LTD must be integrated with an UML modelling tool under the ECLIPSE platform and generated test cases are exported into a test repository and adapted to the test execution platform. LTD supports both the Windows and LINUX platforms.

LTD uses a design model for test generation. LTD uses the UML class diagram and state machine diagram with OCL. UML is used for the graphical model and the OCL code to define the operation in the class diagram. LTD does not include any modeller but it does support the IBM Rational Software Modeller and Borland Together models as input.

LTD splits the test case generation into three parts: preamble, body and postamble. Preamble leads from the initial state to a specific state. Body executes the test itself. Postamble is optional and brings the system back to either the final state of the state machine or to the initial state. From the model LTD automatically generates test case specifications (i.e. the expected behaviour to be tested, which is described as test targets in LTD vocabulary). LTD then automatically generates the preamble, the body call, and, optionally, the postamble from the model. LTD supports a traceability matrix for tracking errors requirements. LTD also provides a timeouts setting. Although LTD does not provide any coverage criteria for generation guiding, coverage metrics are reported.

LTD provides generated tests which are exported to an external tool for managing and execution. LTD provides a large variety of adapters for different exporting formats. The company also promised to create new ones quickly if there were no suitable ones available. Do-it-yourself adapters are also provided.

The test execution framework performs test execution reporting. LTD provides information on test suite coverage and a traceability matrix. The traceability matrix presents dependencies between test cases and requirements.

LTD is highly useful if a company already has it and is familiar with compatible modelling and test execution tools. In such cases LTD can easily be set between the modeller and test execution platform.

5.3 Markov Test Logic

Markov Test Logic (MaTeLo) is a commercial offline model-based tool provided by All4Tec [27]. MaTeLo has its own modeller and the test is exported in textual notation. MaTeLo uses Markov Chain models and therefore focuses on test control oriented systems. MaTeLo provides exporting formats for automatic and manual test execution. MaTeLo only supports the Windows platform.

MaTeLo is divided into two separate programs, Usage Model Editor and Testor. The Usage Model Editor uses the Markov Chain Usage model for modelling notation. It is a finite state machine extended with probability numbers. It is not extended with programming languages but does accept variables, Scilab/scicos functions, and Matlab/Simulink Transfer functions, extending the model for simulating expected results. While the use of these variables and functions limits the model's complexity, they are not as effective as programming languages. The lack of a programming language extension sets limitations on present data-flow models. MaTeLo accepts many kinds of models as inputs via the MaTeLo converter. These inputs are important for the reuse of existing models. While MaTeLo only provides deterministic models, these can include asynchronous inputs. MaTeLo relies on the design model aspect.

MaTeLo Testor takes the model as input and generates a test suite. Testor validates the model before usage. Validation means checking modelling errors like unattainable states. Test generation is able to guide the user with four algorithms: random, boundary value testing, most probable route, state coverage and transition coverage. MaTeLo also provides time limits. Although MaTeLo does not offer any target based test generation possibilities, test generation can be done with probability numbers.

MaTeLo cannot execute tests itself. It is, however, possible to export test suites into HTML, TTCN-3 or TestStand formats. TTCN-3 and TestStand are used for automatic test execution, and HTML for manual testing. TestStand is a test management tool created by National Instruments. During the testing process, Report Management is used for making a report of test campaign monitoring. Report Management has the additional feature of presenting pleasing graphical figures of the testing process.

Probability numbers are the main idea behind the Markov Chain. MaTeLo is therefore a powerful choice for control-oriented testing compared with the test output formats of other tools, which are relatively limited.

5.4 Conformiq Qtronic

Conformiq Qtronic (CQ) is a commercial MBT tool by Conformiq. It provides both online and offline MBT [28]. CQ works on Windows and Linux operating systems. CQ is a general-purpose MBT tool. Thus, the models and test execution techniques and algorithms are not tied to any specific domain or platform. Qtronic provides its own components for modelling and test execution, but it can be integrated with external tools.

CQ has its own modelling tool but it also accepts inputting of models. Qtronic supports multi-threaded concurrent models and testing of non-deterministic systems in online mode. The modelling tool provided by CQ is the Qtronic Modeler and it uses Qtronic Modelling Language (QML). QML means UML statechart extended with java or C# code. Qtronic supports also CQλ and any UML2.0 models as input. CQλ is a variant of LISP. UML2.0 can be used for importing models from third party modelling tools. UML2.0 has to be saved in XMI format before importing. All of these can all be extended with java or with C# in the same manner as with Qtronic Modeler.

Qtronic provides nine sophisticated coverage criteria, which provide good test generation guiding possibilities. In offline generation, Qtronic is able to limit search tree depth and maximum delay for response. In similar fashion to LTD, Qtronic also provides test generation based on specification requirements, which are interpreted and described in the model. CQ also provides manually created use cases for test generation guidance. Qtronic providing coverage criteria are state coverage, transition coverage, 2-transition coverage, implicit consumption, boundary value analysis, branch coverage, method coverage, statement coverage, atomic condition coverage.

In Online, perspective mode the user can choose one of three alternative walking techniques: random, Markov Chain or coverage guided. The Markov Chain algorithm does not promote the same route again which means a wider scope of

walking. The coverage guided walking technique focuses on covering selected coverage criteria. The coverage-guided technique is very useful when the testing time is short. In online mode, the user can also define the maximum latency time. Test execution can optionally be paused automatically, stopped when all coverage criteria are fulfilled, or stopped after a single test run.

In offline mode the user is able to choose look ahead depth and maximum delay time. Look ahead depth controls the amount of CPU time used for planning the test scripts. Maximum delay signifies response waiting time after the sending of input. The offline mode also makes it possible to minimize the size of the test sets, or to generate only finalized test sets.

Adaptation is done by plug-ins: scripter plug-in for offline while MBT and both adapter and logging plug-ins for online testing. The plug-ins can be performed in C++ or Java. The Qtronic package already has some plug-ins, for example TTCN-3 or HTML scripter. It is easy to make a new plug-in for a specific format.

Conformiq Qtronic is a true MBT tool with a very general approach. Open plug-ins makes the tool highly flexible and easily adaptable to different domains. Qtronic is currently the only MBT tool that provides online MBT.

5.5 Reactis

Reactis is a commercial offline model-based testing tool for embedded software. It is specialized in embedded software testing. Reactis is strongly bound with MATLAB and works on the Windows platform [29].

Reactis uses MATLAB / Simulink / Stateflow for modelling. Stateflow is a graphical design tool for design and simulating event-driven systems based on finite-state machine theory. It is focused on modelling embedded systems and therefore Reactis is domain-specific for embedded systems. Reactis is a workable tool for model simulation. For example, the user is able to load a test case and check how it works in the model. Values can also be changed during simulation. This facilitates validation of the model for testing.

Reactis can generate tests in two main ways; randomly or guided with some coverage criteria. Reactis has ten different sophisticated coverage criteria. Random tests and coverage criteria-guided tests can be included in the same test suite. There is also the possibility to download the previous test suites as part of the new one. The amount of generated test cases can be set.

The generated test suite can be exported in several Matlab formats, as a plain ASCII file or in the comma separated value (CSV) format. Matlab formats can be used with MathWorks products and CVS to run the test suite in hardware in the loop environment.

Reactis provides a wealth of information of the generated test suite. Since Reactis does not run any tests, it cannot write a test report.

Reactis is an adequate MBT tool for the embedded software domain. The model can be tested and debugged in detail in Reactis before test generation, which is likely to reduce the number of faults. The test data generated by Reactis can be exported and tested in Matlab.

5.6 Spec Explorer

Spec Explorer is a MBT tool that is allowed for use in any non-commercial purpose. It is made by Microsoft and accepts only Windows as its operating system. It can test in both offline and online approaches. Spec Explorer is strongly tied to Visual Studio. It uses Visual Studio's (VS) formats, and does compilation in VS. [30, 31]

Spec Explorer uses the textual notations: Abstract State Machine Language (AsmL) and Spec# for modelling. ASML is an executable specification language based on the theory of Abstract State Machines. Spec# is an extended version of C#, with extension to support non-null types and checked exceptions. Modelling can be done with text editors or with an integrated graphical editor. Spec Explorer generates visual finite state machine (FSM) from textual notation for illustration.

When there is a requirement to run the test suite automatically against the implementation of the system it was necessary to write an adapter for mapping Spec Explorer and SUT together. The adapter may be written in C# or Visual Basic.

Spec Explorer offers few coverage criteria. The offline approach gives random walk, transition coverage or shortest path algorithm. Online testing works only with randomly walking. There are also some searching algorithms for sharpening test set quality that affect both testing approaches.

Both offline and online testing are executable in Spec Explorer. It is also possible to export offline test suite in xml format or export executable test code in Visual Basic or C# language. Online testing can be started directly from the Spec Explorer and the tool will continue to run test cases against the model until SUT fails or the user stops the execution process. Spec explorer can present a test very illustratively in the model. The selected test is shown as a different colour in the map.

Spec Explorer is most useful when you are familiar with Visual Studio. Unfortunately, it is just for research purposes. Based on this, however, a very sophisticated tool from Microsoft is being developed. Further reading is available from Spec Explorers homepage [31].

5.7 Guidelines for selection

In this section, we aim to give guidelines based on MBT theory and experiences of evaluating tools.

- **Do you need online MBT or is offline MBT enough?** Offline testing provides test script automatically generating from a model to the existing test execution platform. The offline MBT approach is easier to adopt than the online MBT approach because it is closer to the normal script-based testing process. If the infinite test suites and non-deterministic systems testing are needed, this presupposes the need for the online MBT approach. Still, both of these provide several advantages, which are described in chapter 3. Only Conformiq Qtronic and Spec Explorer provide online MBT testing and Spec Explorer is not

for commercial use. Therefore, Qtronic seems to be the only suitable choice when online MBT is needed for industrial applications.

- **Generic or DSM modelling language?** Model languages can be divided into domain specifics and generics based on domain limitation. In general all are suitable for a large proportion of purposes but domain specific ones are more effective inside the domain. Reactis showed the strongest domain limitation of all the evaluated tools. It only works with embedded systems because of modelling languages.
- **What then is the right modelling notation?** The most important thing over a language selection for testing is to think that how well the SUT is able to model using that notation. The model is the only information of system requirements that the MBT tool has. It has been shown that existing tools typically use pre/post models or transition based notation. Pre/post models are most suitable for data-oriented system modelling. Pre/post models examples include the UML state diagram extended with programming language, Spec# and JML. Transition based notation is most feasible for control oriented modelling. An example of this kind of modelling language is a finite state machine (FSM). For example, MaTeLo uses that kind of model. MaTeLo uses FSM with probability numbers and some extensions. Therefore, MaTeLo can handle an expected output, but does not do so well with pro/post models. [3]
- **Design model reuse for testing in MBT tools:** The design model, which is made for implementation, is able to be reused in MBT if the MBT tool uses the design model perspective. Straight reuse means that the model for testing has the same faults as the model for implementation. In practise, there are also problems with formats between modelling tools and the MBT tester. Based on the evaluation, LTD and Conformiq Qtronic use the design modelling aspect. Those also have several ways to use external modelling.
- **What kind of test requirements is needed?** Good test guiding algorithms reduce effort and improve the quality of test cases. MBT tools provide various collections of test guiding algorithms. Those are described in chapter 3.3. The sufficient level of these is hard to determine beforehand. It could be easier to state that almost all the required algorithms are provided in selected tool.

- **How to map MBT generated tests and SUT together?** Because the offline MBT tool normally exports the test suite, it is important that the format is suitable for the test execution platform. Normally, some of the most general formats are provided like XML and TTCN-3. If there is a possibility for self-made plug-ins, the formats are almost unlimited. Online MBT tool have to map straight to SUT. Therefore, there are not normally existing usable solutions, but rather the layer has to be implemented. Online MBT can provide API for making that layer.
- **How can errors be traced?** In offline MBT the traceability knowledge is included in to the test script. Therefore, the main question is how much information the test script has. In online testing, the MBT tool takes care of traceability. Tools also provide do-it-yourself plug-in probabilities. This field is fragmented and tools should be used in real testing in order to see the traceability efficiently. Therefore, we only recommend that the selected tool has adequate traceability features. MBT as a whole is almost useless without traceability.
- **What kind of documents are automatically written?** Offline MBT tools export test suite and test execution platform execute tests and write the report. Report quality depends on script quality and executors features. Online MBT tools write tests and it is useful in such instances to take care that suitable report formats are provided.

6. Case study: Tienoo

The main purpose of this thesis was to describe a case study where model-based testing is used in an agile project. The case study's project name was Tienoo. The purpose of the Tienoo project was to develop a system for the FACMA project while the MBT part is made for this thesis. This chapter reviews the study case.

6.1 Introduction

This case study was performed in cooperation with the Mobile Facility Management Services (FACMA) and Rapid Iterative model driven Testing in Agile context (RITA) projects from VTT Technical Research Centre of Finland, the software project course from Oulu University and SoPro (productivity increasing in software industry) project from Joensuu University. This study case was a RITA and FACMA equal joint venture at the beginning. Then SoPro participated with two persons, who worked all the time in Joensuu.

This thesis is made for RITA. The RITA research area was focused on getting information on new techniques in the agile context. FACMA was the customer of this study case. FACMA needed a demo version of software for trial. SoPro project research areas included pair programming and distributed agile developing process practices.

The Tienoo case study was a distributed agile developing process. Two persons worked in Joensuu University and five people in VTT premises in Oulu. The author was working for the RITA project and his roles were model-based tester and project manager. Four members came from Oulu University who performed a software project course. Each of them had their own area of responsibility, these being tester, metric, architecture and quality measurement. Two persons were working in Joensuu. They were developers without specific responsibilities.

In this case study Conformiq software is used with Conformiq Qtronic as MBT tool. Conformiq Qtronic was chosen based on the evaluation described in chapter 4. The main reason for selection was that we did not have knowledge of

the system domain before the project began. Conformiq Qtronic was the most general tool and therefore had the best adaptability for different domains.

6.2 Developing process

Our team did not have much previous experience of techniques that are used in this project. Only one person had real work experience in java. Other member's java programming experiences came from Universities courses. The Mobile-D™ developing process was also new for all with the exception of one person. We used several other techniques like Java Servlets, JSP, J2ME, Google Map and SVN that no one of us had any working experience of. Members who worked in Oulu participated in Conformiq Qtronic and Mobile-D™ training days before the project started. Mobile-D™ training was organised by VTT and Conformiq Qtronic training was organised by Conformiq.

The Mobile-D™ (see 2.3) working method does not give communication rules except release and planning day meetings but it incites teams to self organize adequate communication. Adequate communication was a big challenge because we were a geographically distributed agile team. Joensuu members only visited Oulu on the first planning day and final release day. We mainly used videoconference, phone calls, Skype video calls and Windows Messenger chat for communication. Videoconference was used in the planning and release meeting. Videoconference was a good way to communicate but there were technical problems several times and organizing it needed effort. Phone calls were used when an immediate connection was needed like those related to solving architecture. Skype video calls were used in wrap-ups. Wrap-ups were kept irregular when it was considered necessary. The advantage of Skype video calls advantages was that they were lighter to organize than videoconference and everyone was able to participate. Skype voice and video quality were poor so we could not use that in planning or release meetings. Windows Messenger chat was used to communicate with small problems between Joensuu and Oulu during the working day. Everyone could take a personal connection with somebody using Messenger. We also used Taskmaster for sharing the knowledge of implementation real-time situation. We used a taskmaster's web site version and physically on the wall. The wall was much more illustrative than the web one but the web site version was easier to share between two places.

We increased communication using pair programming and post iteration workshops. We did not use pair programming all the time, but that much that at least two developers could develop every single part of the whole. We kept that goal relatively well. This way we could assure implementation continuance when someone could not come to work. Post Iteration workshops (PIW) were our style to tell all of our worries. Of course, worries appeared out of the PIWs, but PIW presented a better atmosphere for telling stories. PIW feedback showed that members demonstrated good teamwork and team spirit but worried about technical problems at the beginning of the project. Later most of technical problems had been solved and the biggest worry was communication between Joensuu and Oulu.

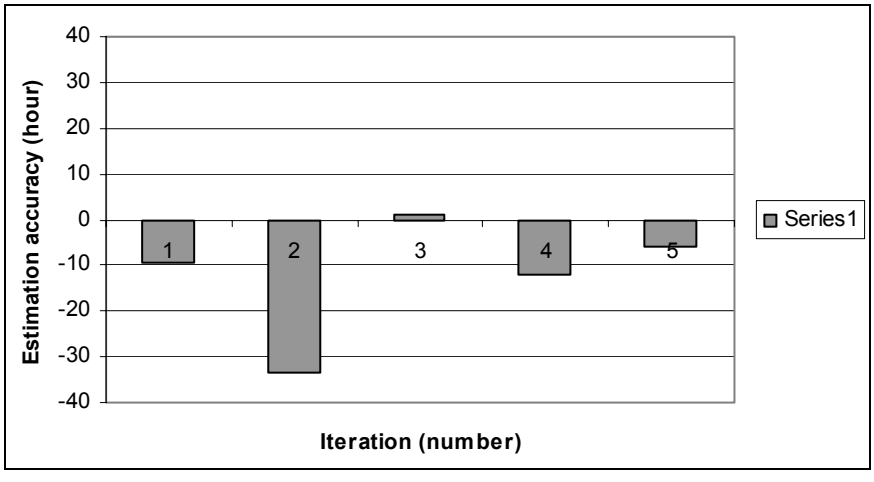


Figure 14. Estimation accuracy in iterations.

In its entirety, the Tienoo project went well. In agile developing processes, estimation accuracy is a good metric to describe process maturity. At the beginning of the project, estimations did not match real time needs. During the process, the team learnt to estimate and as a result estimations became more accurate. In Figure 14 estimation accuracy is described so that real implementation time is deducted from the estimated time. For example, in the first iteration, we used about ten hours more time than we estimated. The second iteration varies as much as the first one because it was three times longer than the first one. In iteration three, the estimation is relatively good. The fourth and fifth iterations took more time because of integration problems. Joensuu and Oulu developed

mobile code too far from each other and their understanding of architecture was relatively different. This led to a big integration problem.

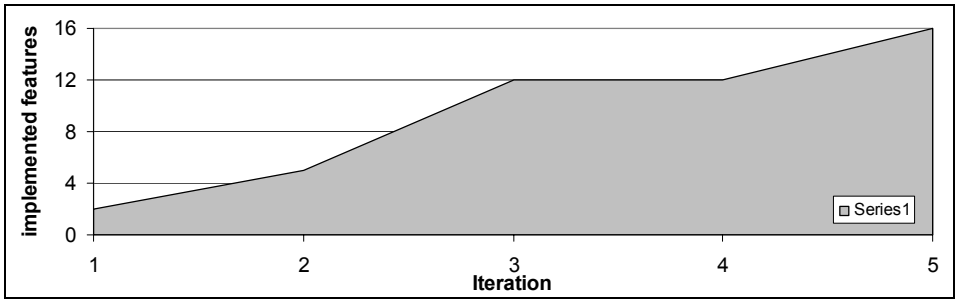


Figure 15. Developing process propagation.

We implemented all the features that were appointed at the beginning of the program. The customer invented more during the project but we had no time to implement those. The process did not proceed without trouble. Figure 15 presents the number of implemented features after all iterations. From first to third iteration, development was increasing and at a fast rate. In the fourth iteration, we noticed the integration problem and we could not integrate all parts for the fourth release. That resulted in failure because we had no new working features for the customer. We integrated every part together and developed minor features in the fifth iteration. In this figure have to notice, that earlier features were smaller as a whole than later. Therefore, the feature number is not comparable in the beginning and end of the project.

6.3 Tienoo software features

At present maintenance men are reporting on the completion of assignments after their rounds. Tienoo software makes this reporting possible immediately after a working situation. The Tienoo application is both a reporting tool and a prescription-giving aid. It helps maintenance men to remember all their designated tasks.

Maintenance men must be able to report their work in real-time on the spot. Reporting has to be as easy as possible because the user focus must be on

working, not on using a mobile application. Reporting can be manually written text, voice or photo. Maintenance men must be able to locate themselves both outside and inside buildings. The system must allow voice mail to be left so that maintenance men are able to listen to it.

A maintenance man logs in the morning by touching a personal identification card with a mobile phone. The personal identification card includes a Near Field Communication (NFC) tag and the phone relays information to the backend system that the person have started work. When he arrives at his workplace, he touches the location tag. The application launches and creates a Bluetooth connection to the GPS locator to get coordinates. When the phone has the coordinates, it sends the coordinates to the backend system. The backend system finds tasks that are determined for the place and sends back a list of tasks. The phone presents tasks to the user. The user does the tasks fills the application and presses the send button. The phone sends report to the back end system. The backend system saves it to the database and shows it in the administrator web page. From the web page a manager sees the task situations in real-time.

Inside the location is made with NFC tags that have place knowledge. Inside tasks for maintenance men are normally conducted on a room level. Therefore, it is enough to use room specific tags. When inside the maintenance man presses a tag with the phone and the application gives a task list.

The maintenance man is able to listen to a voice mail that is directed to that place. He can also write text or take a picture as a report. The implemented solution also includes a web site that enables the management of tasks, persons, places and check reports.

6.4 System implementation

The system includes three parts: a web site for administrator, a mobile application for the employee and a back end system for data handling. Figure 16 describes the high-level overview of system architecture. Appendix 1 shows features of system parts.

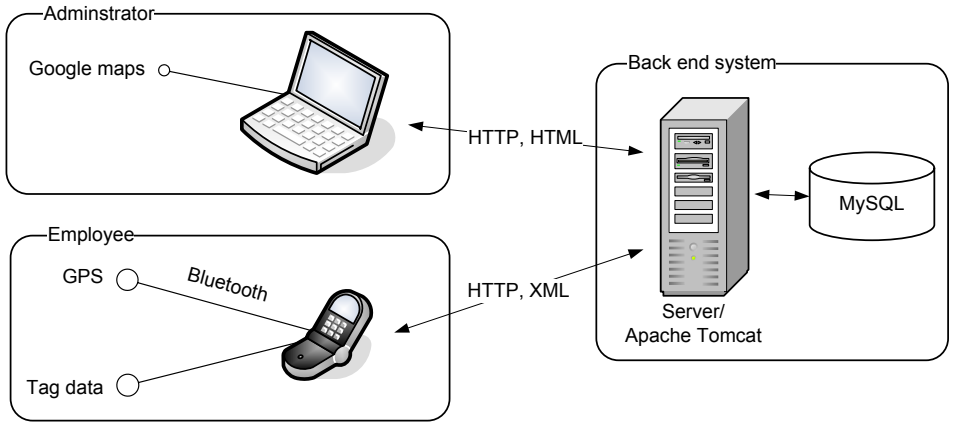


Figure 16. The system overview.

Via the web site, the administrator can manage tasks, employees and places. All those are able to be added, and deleted. Employees and tasks can be added by writing the necessary details and saving those. Place adding has been implemented using the Google maps service. The Google map makes it possible to see an area in a map and click on the corners of the place. The Google map also colours the determined area. It is also possible to bind tasks to the area i. Bounded tasks are sent to the maintenance man when he is located to that area. In the report page there is list of reports with details. There is also a link to the voice file when the maintenance man has given voice mail with his/her report. The web site is running in Ubuntu in an Apache Tomcat server. It has been implemented using Java Servlets, JSP, JavaScript and Google Maps.

The mobile phone application has been made for the usage of maintenance men and is as easy as possible to use. The application is implemented in java using high-level user interface components. Bluetooth was used for making a coordinate query to the GPS block. The NFC technique is used for reading NFC tags. NFC is closely readable RFID technique. The maximum NFC reading length is 10 centimetres. Tags include some data that is transferred to the mobile phone through touch. The data can order the mobile phone to call some number, to go to some web site or launch some program.

6.5 Testing system

The testing system consists of three main parts: a test case generator, test executor and a system under testing. Conformiq Qtronic was chosen to be the test case generator based on previous MBT tools evaluation. Qtronic exports the test suite as a text file. The test suite was imported to the JwebUnit. JwebUnit was the test execution platform that runs test cases against SUT. The overview of the testing environment is presented in Figure 17. Developing the model-based testing system does not take place incrementally but is more continuous and comprehensive in nature. Therefore, the model does not need to be ready before scripter developing. In this case, first, we made a very lightweight model, scripter and test execution platform then executed tests out and ran those. Next, we developed a little bit of every part and tried again. This iterative way kept results testable most of the time and it was then possible to use them as regression tests.

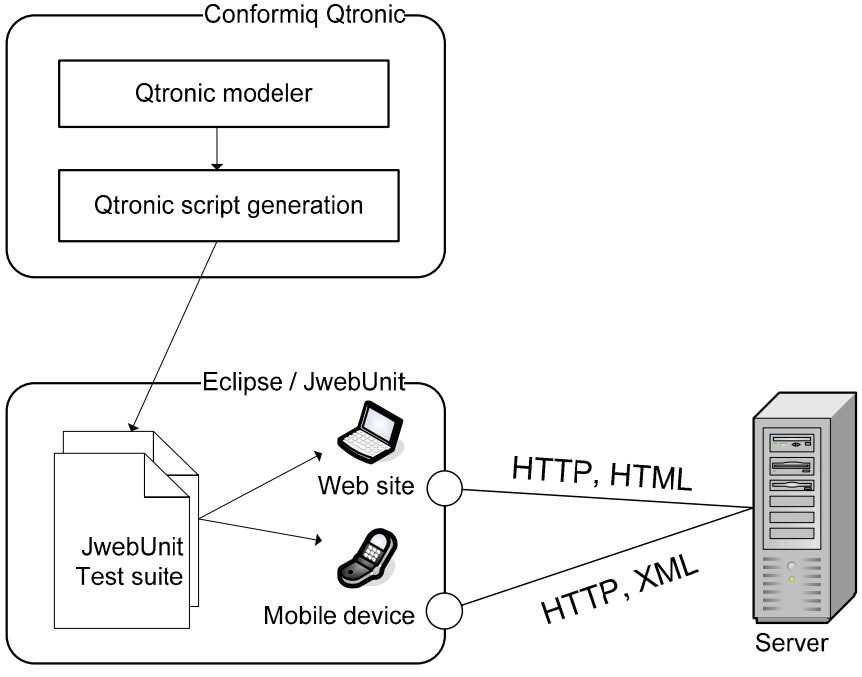


Figure 17. Testing system overview.

```
1   private void testLogin(){
2   beginAt("localhost/login.html");
3   assertTitleEquals("Tienoo");
4   setTextField("username",name);
5   setTextField("password",passwd);
6   submit("", "OK");
7   assertLinkPresent("Kirjaudu ulos");
8   }
```

Figure 18. JWebUnit test case example.

JWebUnit is a java framework that facilitates the creation of acceptance tests for web applications. It is an Eclipse plug-in and therefore works on Eclipse. The JWebUnit user interface is described in Appendix 3. Figure 18 describes an example of a JWebUnit test case. The example is made using methods provided by JWebUnit. The example is a login use case. First the web address: localhost/login.htm is accessed. Then the title is checked and username and password are input to the fields. Then the ok-button is pressed and checked that there is logout-link available. Then the script can be run by pressing the run-button in Eclipse. JWebUnit presents failed cases with red marks, successful marks with green marks and cases which have some errors with dark blue marks. The marks can be seen on the left side of the picture in Appendix 3. [32]

6.5.1 Modelling

Modelling is an important phase of MBT because the model is the MBT testing tool's only knowledge of the SUT. All test cases are generated using this knowledge. Qtronic accepts several languages as input. In this study case, we used Conformiq Qtronic's own tool, Conformiq Qtronic Modeler. The Qtronic model has to be made from an implementation aspect, but at a higher abstraction level than the implementation model. The abstraction level is described in chapter 3.2. A high abstraction level makes the model lighter to develop and easier to understand. The Qtronic modeller uses a state machine that is expanded with Qtronic modelling language (QML) language that is close to java. The state machine is similar to the java program's run method visualization. So part of its functionality can be hidden in text files and in this way the model becomes clearer. The model is also possible to do wholly in text format. QML is

Qtronic's own language and it has java syntax with several limitations and extensions.

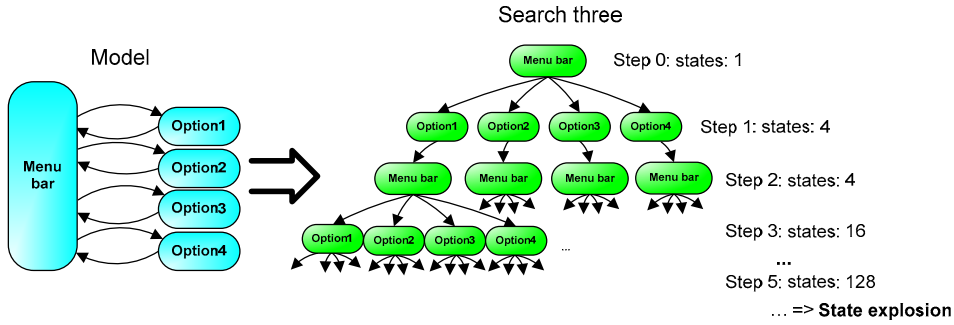


Figure 19. State explosion.

At the beginning of this case, we thought of making a sharp model, which could wholly cover the administrator web site logic. We thought that this approach could give several good test cases for testing system logical working correctness. It could have been a good approach but that was ultimately not successful. The model grew huge in size and complex, because of the website's menu bar. The menu bar made it possible to move to every main topic from every single state. That led to a huge amount of transitions in the model and MBT could not find tests efficiently. MBT tool's test case seeking problem is described in Figure 19. The searching tree grew exponentially and it caused calculation problems. The other problem was that we were not aware of which things have to be put in the scripter and the model. This troublesome modelling aspect added to our novice skills led to the situation where implementation was faster than MBT model developing.

Later we noticed that some of these problems could be solved with a better knowledge of MBT tool algorithms. Therefore, part of the reason for this failure was our lack of skills. We also noticed that for efficient MBT implementation a lot of practice is needed.

When we noticed that the first try was failing, we stopped it and started to make a totally new model. We started the second one from a much higher level. The previous model was not reusability because of the changed modelling aspect. The scripter was fully reusable because it only depended on the test execution

platform's format (See. 3.6) and the test executor was the same. We made the second model according to the requirement level. The requirement level meant here that requirements were able to see straight into the model and there were no more details than the requirements described. The requirement level gave enough of an abstraction level to model system logic.

In this case, the model included two threads: a main system thread and a database knowledge describing thread. The model is shown in Appendix 4. and Appendix 5. The main thread included an idle state where the system starts. From the idle state it is possible to try login from a mobile or from the web site side. If login succeeds in the web site there is a possibility to perform actions in whatever order is required. Mobile side reporting is not meaningful if the user has no other task list to report. Therefore, on the mobile side, the order of actions is limited. All the time the database thread takes care of added and deleted data, so the test case can go in random order in the model and always has the right data. We tested the model behaviour also in making html-format sequence diagrams and by reading required model behaviour from that.

6.5.2 Making tests executable

We used the Eclipse JUnit testing tool with JWebUnit library as the test execution platform. For translating test cases in JWebUnit syntax, we had to make a scripter. In Qtronic, the scripter means a plug-in that is imported to the tool. Qtronic accept java and C++ languages for creating plug-ins. In this case java is used as the programming language in the scripter. You have to implement a class that realizes a super class. The super class determines functions that the Qtronic calls for when it is writing a script. In Figure 20 the skeleton of the scripter that was made for this thesis is presented. Writing type is written in scripter that has been made by java. Therefore, all java-writing styles are possible like write text to the file, the command line in Linux or in the database. Some examples of those functions are presented in Figure 20.

```
1   public class TienooScripter extends ScriptBackend {
2       …
3       public boolean initialize(String args) {…}
4       public boolean beginScript(){…}
5       public boolean beginCase(){…}
6       public boolean testStep(…){…}
7       public boolean endCase(){…}
8       public boolean endScript(){…}
9       public boolean uninitialize(){…}
10      …
11  }
```

Figure 20. Qtronic Scripter example.

Scripter is like an abstraction-growing layer between a model's abstraction state and a test executor's abstraction state. Practically, this means that the scripter has to translate test case knowledge from the model to that test execution syntax. In this case, the model is feature level and the test executor works on a manual level. The manual level means that with JwebUnit tool can go to the web site, click on links, put texts to the text fields and so on. Therefore, it is a big gap from a feature like "Add user" to the manual clicking in web site. We solved this problem using keyword-driven testing principles [33]. It makes abstraction higher, makes the model simple and helps the user to focus on the system logic testing. We needed to test the system's logic from the web site interface, so there was no meaning to test all routes to do that in this testing style. Therefore, we made methods that make further actions and test cases include as little information as possible. Figure 21 presents an example of a test case and a method that is similar to keyword testing using.

```
1       /**
2       * This is auto generated test case number 9
3       */
4       @Test
5       public void testCase9() throws Exception {
6                webLogin("mikkomatti","miksu");
7                TextPresent("User:");
8                addTask(01,"Luutua lattia","1111");
9                deleteTask("Luutua lattia");
10               …
11      }
12      …
13      /*
14      *  This method make (These methods are provided by
    JWebUnit)
15      */
16      private void webLogin(String name, String passwd){
17               beginAt("/login.html");
18               assertTitleEquals("Tienoo");
19               if(!name.equals(""))setTextField("user", name);
20
        if(!passwd.equals(""))setTextField("pass",passwd);
21               submit("", "OK");
22      }
23  }
24  …
```

Figure 21. Test case and method examples.

6.5.3 Tests generation and execution

Qtronic take the model as an input file. Firstly, Qtronic checks the model to ensure that there are no syntax errors or logically impossible things. Qtronic makes it possible to choose search depth, coverage criteria and requirements for guiding the test generation. A search debt defines how deep the searching tree can be. Coverage criteria mean only the required model coverage in generating test cases. Coverage criteria are described better in chapter 3.3. Qtronic's coverage criteria algorithms are very mature, but those all wholly fulfil the required expert stage modelling. Requirements are targets in the model. If the generated test case passes the target current the requirement is fulfilled.

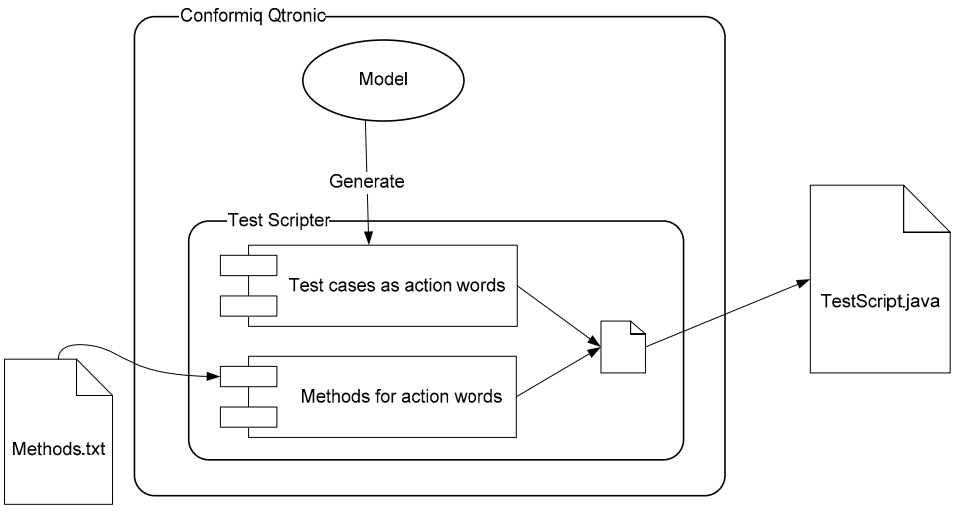


Figure 22. Scripting overview.

The test generation simply required the press of a button and a short wait. The test suite appears in a determined place and format. In this case, we generated tests in a specific folder and in a JwebUnit format. From there we copied the file into the Eclipse JwebUnit testing environment. Eclipse can run tests by pressing a button and it also shows failed test cases. Figure 22 presents an overview of scripting process.

Although Qtronic is the most mature tool according to our experience, it still has some problems. Scripter coding for Qtronic is just like coding the java program and it is possible to do in some developing tool, but model making is not so easy. When some mistake is detected in the model, Qtronic says only that there is an error. It takes a great deal of time to try to find these small mistakes all the time. Conformiq has developed effective algorithms but still there is room for improvement. If the trigger and the conditional statement are apart and there are loops between those, Qtronic cannot find a test case to fulfil the condition statement, or if it does, it takes a very long time. The lack of algorithm, occurs arguably from a loop affected state explosion. String variable testing could also be nice to include in Qtronic. Now it only makes tests with the correct string field value or an empty value.

7. Analysis of results

The main goal of this thesis is the MBT adapting agile process. This chapter reviews the main results of the study case. The hypothesis is presented in chapter 4.

Successfully adapting agile and MBT, model-based testing, for the agile context requires that team members see it as a good thing. The self-organizing team is at the heart of the agile process. Self-organizing means that the team can decide on used practices. If MBT or generally a test engineer in an agile team is not seen as good thing, it can be outsourced. Agile methods only determine positive code testing and there are no rules for making other testing in the project. Therefore, it was an interesting challenge to try to adopt MBT in an agile process. In this chapter our main observations are described.

An MBT testing engineer can work in the developing team and successfully perform MBT in an agile team. We put the MBT in an agile developing team as we described in chapter 4. That was better than we expected. We noticed a number of advantages but only a few problems in MBT including those to the agile team. At the beginning, we had several problems for adapting the MBT tool for JWebUnit tool and run JWebUnit tests against the SUT. The first problem we had to solve by asking from the tool vendor but the second one the developers helped a lot. The JWebUnit tool had several limitations. Thus, there had to be web sites, which were able to be handled with the JWebUnit tool. For testing the server from the mobile point of view, developers made a small web site. They also had to make some structural changes for the administrator web site because of the testing tool. It was also useful to review the MBT model for some developer and in that way be surer that the behaviour was right. Because we made MBT and were developing it in the same time and the same room, the developer knew immediately when the test engineer noticed a fault. Because the test engineer is also a technical person, he asked every time why there was a bug. The developer described the reason and they designed the solution together. There are clear advantages to reveal bugs face to face than keep some bug list in a website. Firstly, the bug is fixed earlier. Secondly, when the test engineer talks about bugs face to face, the developer remembers it better and does not make the same fault so easily again. The only trouble we noticed was the developer's industrial peace: If someone is disturbing you all the time, it is hard to concentrate.

Testing requirements should be handled as normal product requirements in the agile process. In the agile process, low level testing is normally included in tasks and higher level testing is outsourced to a separate testing team. Therefore, there has been no need to handle testing in the agile process. Agile methods have good practices for dividing features for tasks and handling tasks in the process. We used tasks handling practices for testing effort amount handling. In practice, we estimated the time for MBT during the next iteration. This worked well. It might be workable to divide MBT into smaller tasks and try to handle those as developing tasks. This could be the same for testing as task handling is now for developing.

MBT is suitable for pair programming. At the beginning of the MBT development process, we tried to do MBT in the pair programming way. We did not do it much because we had five persons in Oulu and we decided developing pairs to be more important. However, we tried pair programming a couple of times and our investigation showed that it works well. Developers in Oulu had had one day's training for MBT before the pair programming. After training, they became familiar with the tool and with the pair programming method relatively quickly. Pair programming usage ensures that MBT developing does not totally stop if the tester cannot come to work, because developers can also perform MBT.

7.1 Does model-based testing fit in agile iteration?

This study case aims to give knowledge of MBT's suitability for the agile developing process. MBT is able to develop iteratively. Therefore, iterative developing is not the problem, in theory. In practise, the test method has to be lightweight to be able to fit into agile iteration (see chapter 4). Therefore, the question is about the amount of effort required. In this chapter efforts calculations in this case study are presented and the result of the impacts estimated accordingly.

The used metric unit is a working hour and the amounts have been taken from our working diaries. Calculated working hours are true working time and project managing or document writing times are not included. It must be noted that one person was making MBT and the rest of the group was making the implementation part. The MBT test engineer also had project manager responsibility. From Table 1 it can be seen that the first attempt to do MBT needed almost as much time as implementing those features. One reason was that the MBT test engineer

was not familiar with MBT. The bigger reason was that the MBT model was at too low an abstraction level. We tried to present the web site structure in its entirety. There was a link bar on the left side of the site and that increased the transition number by a huge factor. A voluminous amount of transitions made the model unclear and it was difficult for Qtronic to find enough test cases in a practical time limit (see 6.5.1). The model's unclear structure increased work time and we decided to discard it and start a new one.

Table 1. MBT times vs. implementation time.

	Modeling	Scripter	Implementation	MBT/Imp.
First MBT	15 h	41 h	58 h	**95%**
Second MBT	27 h	12 h	310 h	**12%**

The second attempt at performing MBT started from the hypothesis that it was most important to test the system's core logic instead of a web site linking all combinations. This modelling approach is described in Appendix 4 and Appendix 5. The new modelling style reduced effort greatly. We could reuse the scripter plug-in because the test execution platform was the same. Of course, we had to develop the scripter because it was not ready after the first try. Table 1 presents much better efficiency such that MBT only took 12% compared to the implementation time. If the number is the same in the long run, one test engineer can do MBT in eight developer groups. This is a small project and a program and can only prove that this is possible in this context. MBT has provided advantages in the long run [3], therefore it could be able to adopt MBT in an agile context, also in bigger projects.

7.2 Model-based testing vs. script based testing in agile process

MBT is more comfortable for technical persons. We did script-based testing, as described by agile methods and MBT testing in our own way. We noticed that MBT is more comfortable for technical persons than script based testing. MBT tests system development, the heart of which is the MBT test generator. The

better the testing system you build the more it saves you effort later. That poses several challenges and makes it more interesting than script-based testing. Script-based testing is just writing test cases, which means input and output value lists. Script writing does not give true challenges and technical persons like us consider it boring. Agile methods facilitate script writing efforts by sharing the test writing for every developer. There can still be problems with test coverage and the amount of test cases, because tests are written by developers and not a test engineer.

The test engineer must be technical person in order to successfully implement MBT. In MBT it is necessary to build a model and an adaptation layer. In practise, both of these are coding efforts and preclude the intervention of a non-technical person.

MBT generating scripts can also be directly compared to scripted testing. In this case, Qtronic normally made from ten to twenty test cases, which mean 200 lines of code on average. That much code writing manually takes about ten working hours, because the code is relatively simple. Using the MBT way required 39 working hours. This means that manual test case writing is about five times faster than MBT. In MBT, a computer is designing test cases so it is hard to say whether those are better test cases than those which are manually written.

7.3 Summary

This study case indicates that MBT adapts to the agile context very well. Our investigations demonstrated that a test engineer working in an agile team is not a problem. MBT does not need too much effort, thus it can be fitted into iteration. The pair programming method can be used for MBT. An MBT maker must be a technical person, but for him/her it is more comfortable than script-based testing.

These results are based on a single case study and most of the team members were novices. Therefore, straight generalisations are not possible from these results. This is still assuming that near beginners are able to do MBT and MBT can be performed inside the agile team. Future research needs to estimate this efficiently and the business of MBT working in an agile project in the long run and in bigger projects.

8. Conclusions

The main purposes of this thesis were to find suitable guidelines for model-based testing tool selection and to evaluate how model-based testing can be adapted to the agile context.

Work was started by presenting the motivation for the topic, after which an introduction to agile developing methods and model-based testing was presented. Actual work made by the author commenced by gathering guidelines for model-based testing tool selection. Gathered requirements were fulfilled by evaluating available model-based testing tools. The applicability of model-based testing for the agile developing method was evaluated in the case study, where the selected model-based testing tool was used for testing a server application, which was made in an agile project.

The results gained from the case study were encouraging. It was possible to do model-based testing in an agile developing team and calculations seemed to indicate that it could be even more effective in future. There was only one case study and most project members were novices, thus the results cannot be considered reliable in a larger context.

References

[1] Naur, P. & Randell, B. (1968) Software Engineering: Report of a conference. In: The NATO software engineering conference1968, October 7–11, Garmisch, Germany. URL: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.pdf.

[2] Salo, O. (2006) Enabling Software Process Improvement in Agile Software Development Teams and Organisations. Espoo, VTT. (VTT Publications 618.) 149 p. + app. 96 p. ISBN 951-38-6869-9; 951-38-6870-2. URL: http://www.vtt.fi/inf/pdf/publications/2006/P618.pdf.

[3] Utting, M. & Legeard, B. (2007) Practical Model Based Testing: A Tools Approach. Morgan Kaufmann, San Francisco. 433 p.

[4] Craggs, I., Sardis, M. & Heuillard, T. (2003) AGEDIS Case Studies: Model-based Testing in Industry. In: 1st European Conf. on Model Driven Softw, December, Nuremberg, Germany. Pp. 106–117. URL: http://www.agedis.de/documents/AGEDIS%20in%20Industry.pdf.

[5] Becker, P. (read 21.5.2008) Model-based testing helps sun microsystems remove software defects. URL: http://articles.techrepublic.com.com/5100-22-1064538.html.

[6] Hartmann, J., Imoberdorf, C. & Meisinger, M. (2000) UML-based integration testing. In: ISSTA ’00: Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis, August 21–23, Portland, Oregon, United States. Pp. 60–70. URL: http://doi.acm.org/10.1145/347324.348872.

[7] Jéron, T. & Morel, P. (1999) Test generation derived from model-checking. In: CAV ’99: Proceedings of the 11th International Conference on Computer Aided Verification, July 6–10, Trento, Italy. Pp. 108–121.

[8] Offutt, J. & Abdurazik, A. (1999) Generating tests from UML specifications In: Second International Conference on the Unified Modeling Language (UML99), October 28–30, Fort Collins, USA. Pp. 416–429.

[9] Manifesto for agile software development (31.8.2007) URL: http://agilemanifesto.org/.

[10] Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002) Agile software development methods. Review and analysis. Espoo, VTT. 107 p. (VTT Publications 478.) ISBN 951-38-6009-4; 951-38-6010-8. URL: http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf.

[11] Ihme, T. & Abrahamsson, P. (2005) Agile Architecting: The Use of Architectural Patterns in Mobile Java Applications. In: International Journal of Agile Manufacturing, Vol. 8, pp. 97–112

[12] Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P. & Salo, O. (2004) Mobile-D: An agile approach for mobile application development. In: OOPSLA '04: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, October 24–28, Vancouver, BC, Canada.

[13] Electronics – AGILE – agile software technologies. (2007, 16.8.2007) URL: http://www agile.vtt.fi.

[14] Robinson, H. (2004) Obstacles and opportunities for model-based testing in an industrial software environment. 1st European Conference on Model Driven Software Engineering, December 2003, Nuremberg, Germany. Pp. 118–127. URL: http://www.geocities.com/harry_robinson_testing/ ObstaclesAndOpportunities.pdf.

[15] Utting, M. (2005) Position paper: Model-based testing. In: Verified Software: Theories, Tools, Experiments, October 10–13, Zürich. URL: http://www.cs.waikato.ac.nz/~marku/papers/utting_mbt_position.pdf.

[16] Schulz, S., Honkola, J. & Huima, A. (2007) Towards model-based testing with architecture models. In: 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, March 26–29, Tucson, AZ, U.S.A.

[17] Hartman, A., Katara, M. & Olvovsky, S. (2006) Choosing a test modeling language: A survey. In: Proceedings of the Haifa Verification Conference 2006, October 2006, IBM Haifa Labs, Haifa, Israel. Pp. 204–218. URL: http://www.cs.tut.fi/~clark/doc/HVC'06survey-preliminary.pdf.

[18] Utting, M., Pretschner, A. & Legeard, B. (2006) A taxonomy of model-based testing, Working Paper: 04/2006. URL: http://www.cs.waikato.ac.nz/pubs/wp/2006/uow-cs-wp-2006-04.pdf.

[19] Huima, A. (2007, 17.07.2007). Model driven testing – the weblog. URL: http://www.conformiq.com/blog/.

[20] Vain, J., Raiend, K., Kull, A. & Ernits, J. (2007) Synthesis of test purpose directed reactive planning tester for nondeterministic systems. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, November 5–9, Atlanta, Georgia, USA.

[21] Kervinen, A., Maunumaa, M., Pääkkönen, T. & Katara, M. (2006) Model-based testing through a GUI. In: Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software, July 2006, Edinburgh, Scotland, UK. Pp. 16–32.

[22] Hartman, A. (31.3.2004) AGEDIS project final report. URL: http://www.agedis.de/documents/FinalPublicReport%28D1.6%29.PDF

[23] Abrahamsson, P., Warsta, J., Siponen, M. & Ronkainen, J. (2003) New directions on agile methods: comparative analysis. In: 25th International Conference on Software Engineering, May 3–10, Portland, OR, USA.

[24] Talby, D., Hazzan, O., Dubinsky, Y. & Keren, A. (2006) Agile software testing in a large-scale project. IEEE Software 23, pp. 30–37. URL: http://www.cs.huji.ac.il/~davidt/papers/Agile_Testing_IEEESoftware06.pdf.

[25] T-VEC: Products (29.8.2007) URL: http://www.t-vec.com/solutions/products.php.

[26] Leirios – SMART TESTING™: Streamline the test design and improve productivity (11.3.2008) URL: http://www.leirios.com/.

[27] All4Tec – home (11.3.2008) URL: http://www.all4tec.net/.

[28] Conformiq qtronic (11.3.2008) URL: http://www.conformiq.com/qtronic.php.

[29] Reactis: Model-based testing and validation (11.3.2008) URL: http://www.reactive-systems.com/.

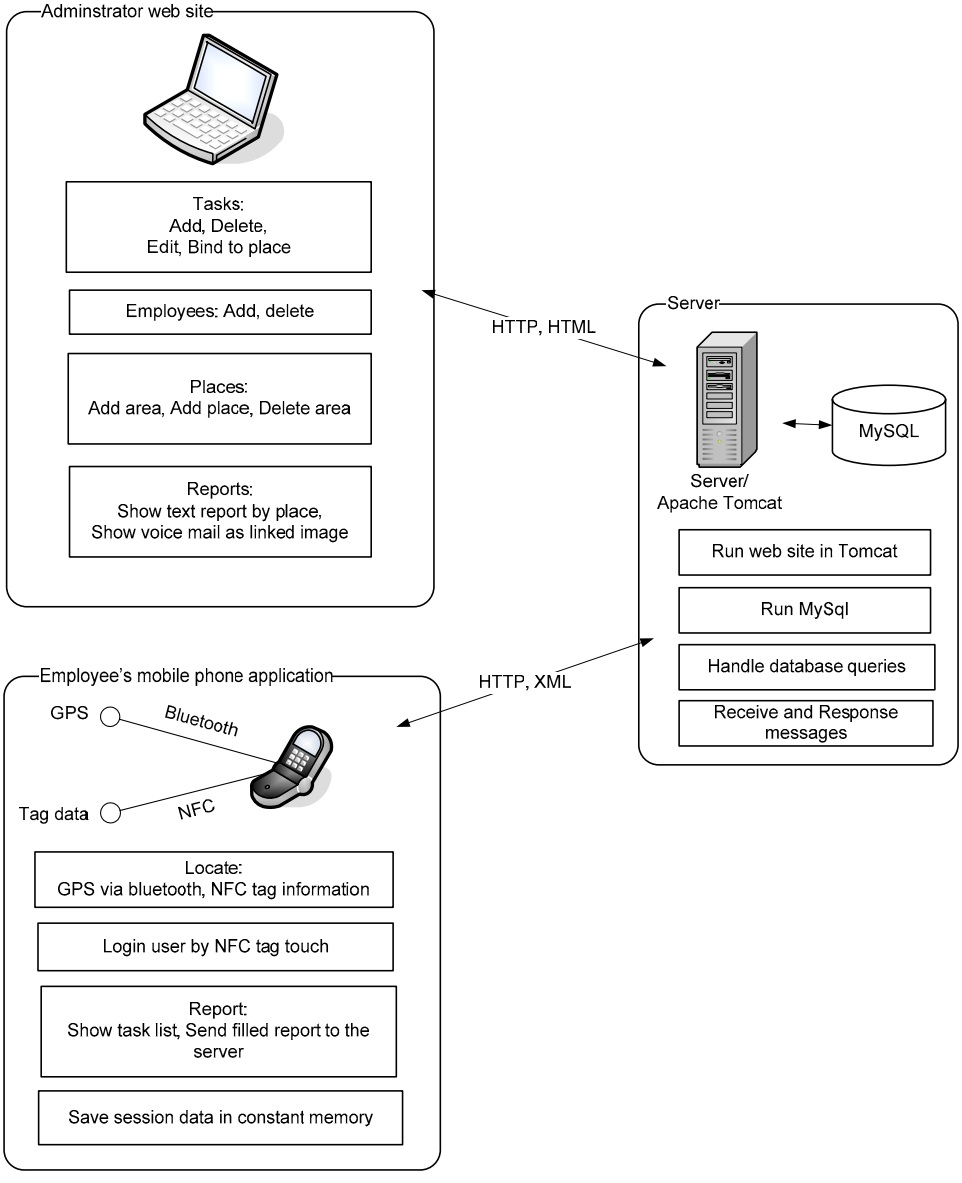
[30] Veanes, M., Campbell, C., Schulte, W. & Tillmann, N. (2005) Online testing with model programs. In: SIGSOFT Softw. Eng. Notes, Vol. 30, pp. 273–282.

[31] Spec explorer – home (13.9.2007) URL: http://research.microsoft.com/projects/SpecExplorer/.

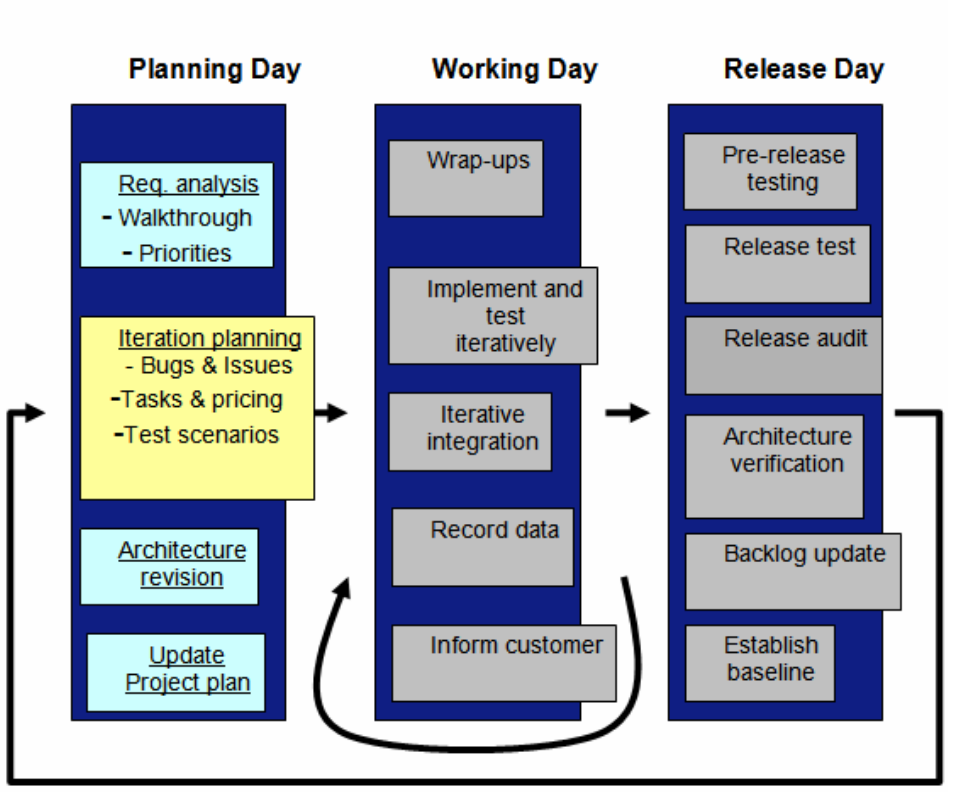
[32] JWebUnit – JWebUnit 1.x (11/04/2008) URL: http://jwebunit.sourceforge.net/.

[33] Katara, M., Kervinen, A., Maunumaa, M., Pääkkönen, T. & Satama, M. (2006) Towards deploying model-based testing with a domain-specific modeling approach. In: Proceedings of TAIC PART – Testing: Academic & Industrial Conference, August 2006, Windsor, UK. Pp. 81–89.

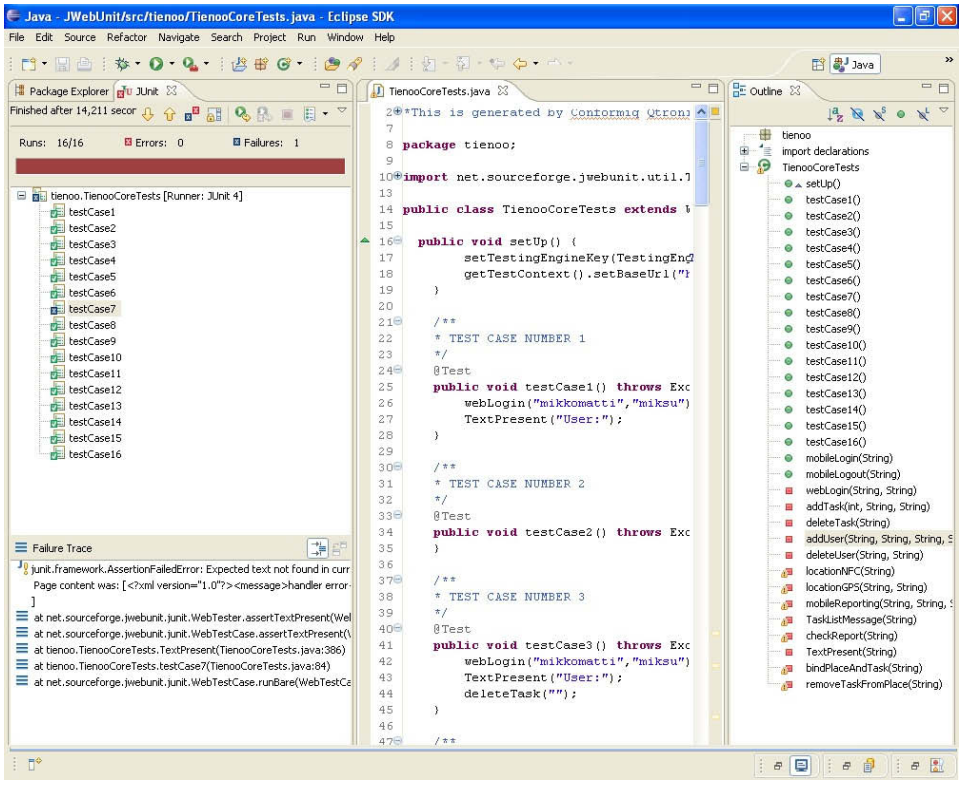
Appendix 1: Tienoo system features



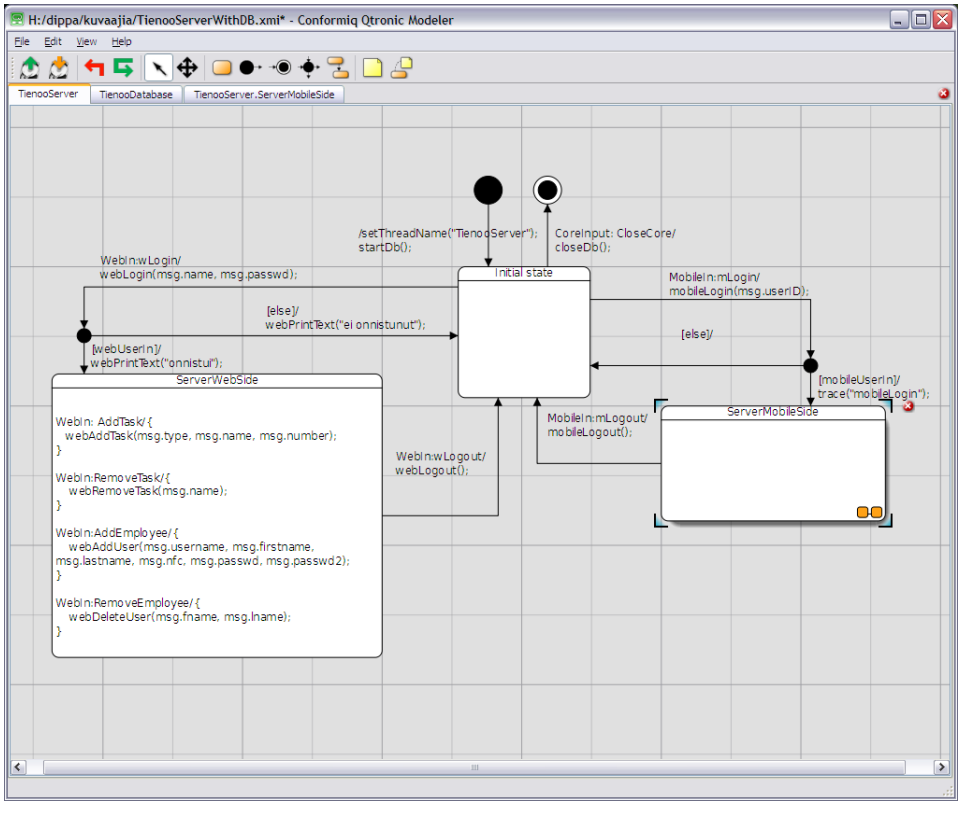
Appendix 2: Mobile-D™ developing iteration

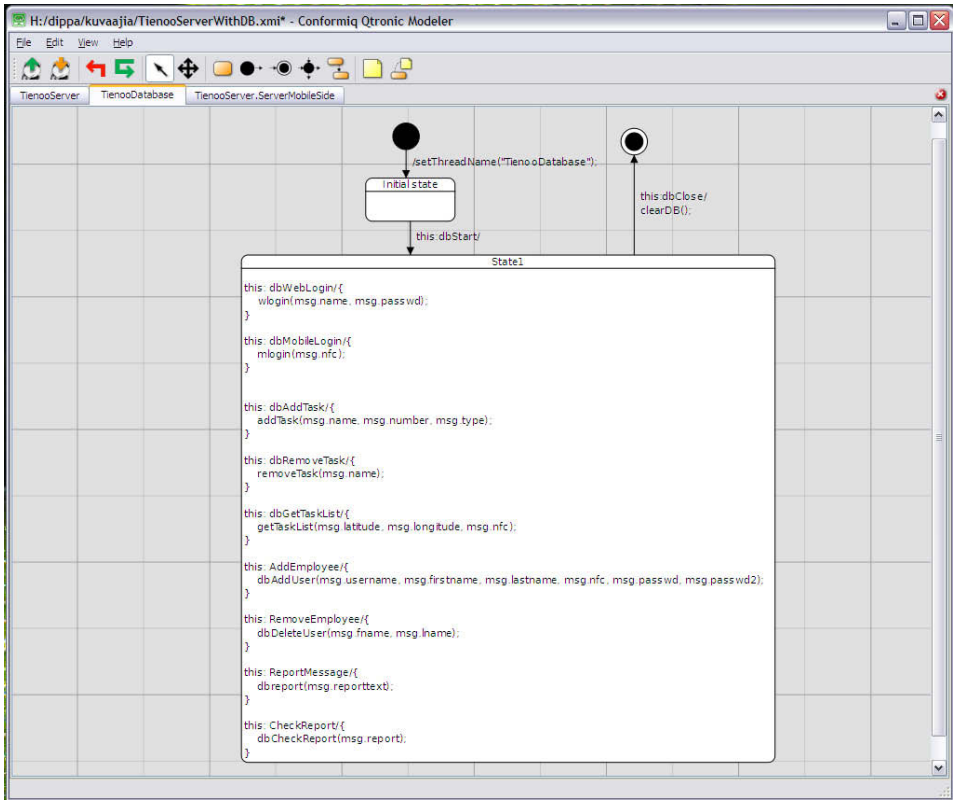


Appendix 3: JWebUnit user interface

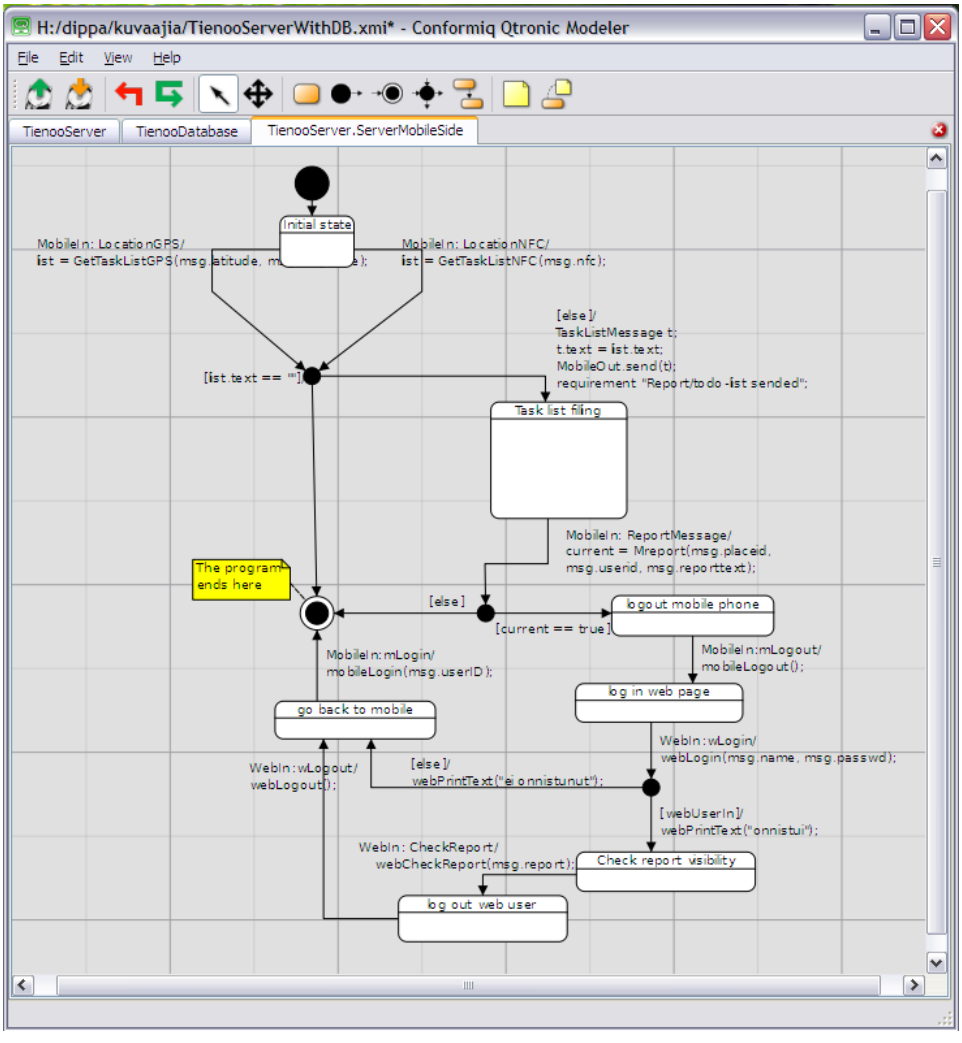


Appendix 4: Conformiq Qtronic Modeler figures: TienooCore and TienooDatabase




H:/dippa/kuvaajia/TienooServerWithDB.xmi* - Conformiq Qtronic Modeler
File Edit View Help
TienooServer
TienooDatabase
TienooServer.ServerMobileSide
/setThreadName("TienooDatabase");
Initial state
this:dbStart/
this:dbClose/
clearDB();
State1
this: dbWebLogin/{
wlogin(msg.name, msg.passwd);
}
this: dbMobileLogin/{
mlogin(msg.nfc);
}
this: dbAddTask/{
addTask(msg.name, msg.number, msg.type);
}
this: dbRemoveTask/{
removeTask(msg.name);
}
this: dbGetTaskList/{
getTaskList(msg.latitude, msg.longitude, msg.nfc);
}
this: AddEmployee/{
dbAddUser(msg.username, msg.firstname, msg.lastname, msg.nfc, msg.passwd, msg.passwd2);
}
this: RemoveEmployee/{
dbDeleteUser(msg.fname, msg.lname);
}
this: ReportMessage/{
dbreport(msg.reporttext);
}
this: CheckReport/{
dbCheckReport(msg.report);
}

Appendix 5: Conformiq Qtronic Modeler figures: TienooServer.ServerMobileSide



Series title, number and
report code of publication

VTT Publications 694
VTT-PUBS-694

Author(s)
Puolitaival, Olli-Pekka

Title

Adapting model-based testing to agile context

Abstract

This study concentrates on model-based testing in agile software developing context. Model-based testing is a software testing technique in which tests are generated from a model. Test can be executed separately later or in motion during the generation. Special focus is placed on examining the adaptability of model-based testing to agile software developing context.

The purposes of this study were to find guidelines for model-based testing tool selection and to evaluate most suitable tool in agile context in case study. First was performed literature survey, where found criteria for model-based testing tools selection. Based on literature survey, was analyzed available tools carefully. Based on literature review and evaluation was made a collection of guidelines for tool selection and selected one tool for case study.

The case study aims to evaluate model-based testing suitability for agile developing project. This case study had two purposes: the first goal was to present model-based testing usage in agile process, and the second goal was to evaluate model-based testing suitability in agile context. Based on empirical findings, it was concluded that model-based testing can be performed in agile process.

ISBN
978-951-38-7119-2 (soft back ed.)
978-951-38-7120-8 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN	Project number
VTT Publications 1235-0621 (soft back ed.) 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)	11375

Date	Language	Pages
September 2008	English, Finnish abstr.	69 p. + app. 6 p.

Name of project	Commissioned by
RITA (Rapid, Iterative, model driven Testing in Agile context)	

Keywords	Publisher
software testing, testing automation, software developing	VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374

Julkaisun sarja, numero ja raporttikoodi

VTT Publications 694
VTT-PUBS-694

Tekijä(t)
Puolitaival, Olli-Pekka

Nimeke

Mallipohjaisen testauksen soveltaminen ketterässä ohjelmistokehityksessä

Tiivistelmä

Tässä työssä käsitellään mallipohjaista testausta ketterässä ohjelmistokehitysympäristössä. Mallipohjaisella testauksella tarkoitetaan tekniikkaa, jossa mallista tuotetaan testejä. Testit voidaan ajaa myöhemmin erikseen tai testata ohjelmaa sitä mukaa, kun testejä generoidaan. Työssä keskitytään tutkimaan mallipohjaisen ohjelmoinnin soveltuvuutta ketterään ohjelmistokehitykseen.

Työn tarkoituksena oli sekä etsiä suuntaviivoja mallipohjaisen testaustyökalun valintaan että tehdä tapaustutkimus parhaaksi valitun työkalun käytöstä ketterässä projektissa. Ensiksi suoritettiin kirjallisuuskatsaus, jossa etsittiin kriteerejä mallipohjaisten testaustyökalujen valintaan. Kirjallisuuskatsauksen perusteella analysoitiin saatavilla olevat olennaisimmat mallipohjaiset työkalut huolellisesti. Analyysin ja kirjallisuuskatsauksen perusteella tehtiin kokoelma suuntaviivoja mallipohjaisen työkalun valinnan tueksi ja valittiin yksi työkalu tapaustutkimusta varten.

Tapaustutkimuksen tarkoitus oli arvioida mallipohjaisen testauksen soveltuvuutta ketterään ohjelmistokehitykseen. Arvioinnilla oli kaksi päämäärää: kuvata mallipohjaisen testauksen käyttöä käytännössä ketterässä projektissa sekä arvioida mallipohjaisen testauksen soveltuvuutta tähän ympäristöön. Tapaustutkimuksen perusteella ketterässä ohjelmistokehitysprosessissa voidaan tehdä mallipohjaista testausta.

ISBN
978-951-38-7119-2 (nid.)
978-951-38-7120-8 (URL: http://www.vtt.fi/publications/index.jsp)

Avainnimeke ja ISSN	Projektinumero
VTT Publications 1235-0621 (nid.) 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)	11375

Julkaisuaika	Kieli	Sivuja
Syyskuu 2008	Englanti, suom. tiiv.	69 s. + liitt. 6 s.

Projektin nimi	Toimeksiantaja(t)
RITA (Rapid, Iterative, model driven Testing in Agile context)	

Avainsanat	Julkaisija
software testing, testing automation, software developing	VTT PL 1000, 02044 VTT Puh. 020 722 4520 Faksi 020 722 4374

VTT PUBLICATIONS

678 FUSION Yearbook. Association Euratom-Tekes. Annual Report 2007. Eds. by Seppo Karttunen & Markus Nora. 2008. 136 p. + app. 14 p.

679 Salusjärvi, Laura. Transcriptome and proteome analysis of xylose-metabolising *Saccharomyces cerevisiae*. 2008. 103 p. + app. 164 p.

680 Sivonen, Sanna. Domain-specific modelling language and code generator for developing repository-based Eclipse plug-ins. 2008. 89 p.

681 Kallio, Katri. Tutkimusorganisaation oppiminen kehittävän vaikuttavuusarvioinnin prosessissa. Osallistujien, johdon ja menetelmän kehittäjän käsityksiä prosessin aikaansaamasta oppimisesta. 2008. 149 s. + liitt. 8 s.

682 Kurkela, Esa, Simell, Pekka, McKeough, Paterson & Kurkela, Minna. Synteesikaasun ja puhtaan polttokaasun valmistus. 2008. 54 s. + liitt. 5 s.

683 Hostikka, Simo. Development of fire simulation models for radiative heat transfer and probabilistic risk assessment. 2008. 103 p. + app. 82 p.

684 Hiltunen, Jussi. Microstructure and superlattice effects on the optical properties of ferroelectric thin films. 2008. 82 p. + app. 42 p.

685 Miettinen, Tuukka. Resource monitoring and visualization of OSGi-based software components. 2008. 107 p. + app. 3 p.

686 Hanhijärvi, Antti & Ranta-Maunus, Alpo. Development of strength grading of timber using combined measurement techniques. Report of the Combigrade-project – phase 2. 2008. 55 p.

687 Mirianon, Florian, Fortino, Stefania & Toratti, Tomi. A method to model wood by using ABAQUS finite element software. Part 1. Constitutive model and computational details. 2008. 51 p.

688 Hirvonen, Mervi. Performance enhancement of small antennas and applications in RFID. 2008. 45 p. + app. 57 p.

689 Setälä, Harri. Regio- and stereoselectivity of oxidative coupling reactions of phenols. Spirodienones as construction units in lignin. 2008. 104 p. + app. 38 p.

690 Mirianon, Florian, Fortino, Stefania & Toratti, Tomi. A method to model wood by using ABAQUS finite element software. Part 2. Application to dowel type connections. 2008. 55 p. + app. 3 p.

691 Räty, Tomi. Architectural Improvements for Mobile Ubiquitous Surveillance Systems. 2008. 106 p. + app. 55 p.

692 Keränen, Kimmo. Photonic module integration based on silicon, ceramic and plastic technologies. 2008. 101 p. + app.

693 Selinheimo, Emilia. Tyrosinase and laccase as novel crosslinking tools for food biopolymers. 2008. 114 p. + app. 62 p.

694 Puolitaival, Olli-Pekka. Adapting model-based testing to agile context. 2008. 69 p. + app. 6 p.

Julkaisu on saatavana	Publikationen distribueras av	This publication is available from
VTT PL 1000 02044 VTT Puh. 020 722 4520 http://www.vtt.fi	VTT PB 1000 02044 VTT Tel. 020 722 4520 http://www.vtt.fi	VTT P.O. Box 1000 FI-02044 VTT, Finland Phone internat. + 358 20 722 4520 http://www.vtt.fi

ISBN 978-951-38-7119-2 (soft back ed.)
ISSN 1235-0621 (soft back ed.)

ISBN 978-951-38-7120-8 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1455-0849 (URL: http://www.vtt.fi/publications/index.jsp)