Model-Based Testing and Defect Distribution per Testing Phase

Erind Pepi

Vårterminen 2014

Handledare
Hugo Quisbert

Informatik/Systemvetenskapliga programmet
Institutionen för ekonomisk och industriell utveckling
Abstract

Automated software testing (AST) is a method that helps in the testing process, by means of time and costs and therefore efficiency. It is, however, not fully automatic, as manual effort is needed prior to execution of these tests. AST has consequently some drawbacks. Worth to be mentioned are difficulties in execution of the tests and test coverage but the main problem is the maintenance of the test scripts, which is reasonably expensive. A newer approach in AST is Model-Based Testing (MBT) that should provide easier and less expensive script maintenance together with better test coverage. However, there are defect types, which cannot be prevented with this model either thus making the question at issue for this paper related to the most problematic phases in MBT, which account for the defects that show after implementation of the software.

Interviews at Ericsson and Spotify (both testing their products with MBT) have shown that the companies have different approaches to MBT. Ericsson uses it simply to generate the test scripts for later offline execution while Spotify has an online approach. These approaches lead to different results since Ericsson does not involve MBT in the test execution phase. Besides the different approaches, it showed that MBT has its own weak points which might lead to defects such as right choice of the abstraction layer and the selection strategy for choosing the generated scripts.
7 Personal Reflections and Future Research ................................................................. 41

Reference list .................................................................................................................. 43
Acknowledgements

I would like to thank first of all my supervisor Hugo Quisbert for his support and very good supervision throughout all these months (even when he was on holidays!). His help is much appreciated.

Big thank you also to my interview persons at Ericsson: Amir Amirkhani and Håkan Fredriksson. They welcomed me from the first e-mail I sent them and continued to answer my question even after the interview, through e-mail communications.

Another thank you goes to Kristian Karl at Spotify who took the time to explain the MBT approach at Spotify and answer my questions.

Erind Pepi,
November 2013
1 Introduction

Much effort is done from software developing companies and their programmers to write error-free code. However it is widely accepted that no software will be delivered to the customer right after the development phase without any testing at all. Automated testing, similar to manual testing must take place before delivery and early in the development phases in order to minimize as much as possible time and costs for correcting the incorrect code. A new approach in testing is Model-Based Testing (MBT). Test cases can be generated through models which later can generate test cases. MBT is suited very well for automation. It has better coverage of the system compared to traditional AST because the test cases are generated automatically from the computer and the MBT tool in use (Binder, 2011, p. 31). This thesis will focus on MBT and the defects after deployment of the software.

1.1 Topic Briefing

Automated testing is not a new topic at all, nor is it a new method for testing. Nowadays there are even jobs as a test developer. Many questions regarding this topic have been already answered. Questions already answered in this area are: the economical gains, the decision to automating manual tests, the lifecycle model (though there is a little debate here on the standard model) and the problems that occur from test tools. Going in deeper in AST, I could find new scientific articles on MBT as an issue solver regarding test maintenance – something hard to achieve with traditional AST. As a relatively new way of testing, there are some open questions in this area. Some of these are:

- What are the types of defects detected using a MBT approach?
- What are the types of defects not detected using a MBT approach but detected after the deployment?
- How is the defect distribution per phase or per type of defects?
- What are the origins of the defects (Limitation of the model? Testing criteria? Tool? Testing process?) (Veer Hooda, 2013, p.543)
One of the above questions regards the defect types that are not detected by MBT and the test cases generated by its tool and I chose to research on that particular one. I chose this question because the main goal of testing is to capture errors before delivery of the software, but still even with a believably better testing technique, there are errors which make it all the way to the customer, thus sometimes questioning the efficiency of this technique. For the time being, there are some small articles or parts of them, which write shortly about defect types and their cause in relation to AST phases. I could find it as an open question in one the latest articles from International Journal of Latest Research in Science and Technology.

1.2 Research Question

The main topic of this thesis will be Model-Based Testing and its efficiency as a testing technique in capturing defects before deployment of the software. As a new and non-mature testing technique, but suggested for better error prevention, it still sometimes fails to discover errors before deployment of the software. Thus making the question at issue of this paper the following:

- Taking into consideration MBT phases, what is the relation between defects, their causes and their distribution per phase?

1.3 Target group

This paper is targeted to students studying informatics related programs, with previous knowledge in programming or AST, that want to obtain knowledge of AST and MBT and want to develop their knowledge about AST and MBT. This paper targets also people and organizations working with MBT. For students with basic knowledge in programming or AST, I will be covering the basics of AST and MBT, which should make it easier to comprehend the paper.
1.4 Delimitations

Focus of this paper will only be on describing AST and MBT from a technical point of view together with identifying the most problematic phases in MBT where most of errors occur. Main focus is on out-of-sprint defects as stated in the research question section since many companies today work with agile methodologies. In-sprint defects are rather difficult to trace since they are solved in-sprint and not documented.

1.5 Previous experiences with AST

My own experiences with AST have been during a six-month internship period at the Enterprise System developer Industrial and Financial Systems (IFS) in Linköping. I have handled (updated and created) several Quality Assurance (QA) related scripts and implemented them in Jenkins, a continuous integration tool. After that period I work as a test automation developer for five months at Wisi Norden AB in Linköping where I created from scratch test scripts that tested the functionality of an IP (Internet Protocol)-TV box. The main function of the boxes was to convert analog satellite or terrestrial signal to IP streaming, together with many other functions. During these two experiences I have only been in touch with AST and not MBT but have seen some flaws in AST like for example the monotonous process of updating the test scripts.
2 Method

In this chapter I will give an overview of methods and the ones I have used in this study.

2.1 Research methods

There are several ways to categorize research methods but there is one main category and that is the distinction between qualitative and quantitative methods (Myers, 1997). The debates on whether which one should be used has been hot and only in the middle of the 19th century were qualitative methods seen as effective and gained popularity among researchers (Ahrne & Svensson, 2011, p.13). **Quantitative research** one was widely used in natural sciences and has different forms of conduct such as: experiments, formal methods, survey methods and other methods that are numerical. **Qualitative research** was developed mostly from social sciences in order to study social and cultural phenomena. Good examples of qualitative research are case studies, ethnography, action research etc (Myers, 1997). When conducting quantitative research it is argued that the point of view of the participants is lost therefore the need for qualitative research methods rises.

A researcher can choose to use the first method, the second or a combination of the two. Choice of which one to use depends on the research questions, traditions and the available resources the researcher has (Ahrne & Svensson, 2011, p.16).

2.2 Philosophical perspectives

Whether conducting a quantitative or qualitative research, both methods are based on some underlying assumptions. It is important to know what these assumptions are in order to continue on with the research. As suggested by Orlikowski and Baroudi (1991), there are three categories that are based on underlying research epistemology: positivist, critical and interpretive. All three can be used in qualitative research (Myers, 1997).

- **Positivist research** assumes that reality can be objectively described by measurable properties that are independent of the researcher. Information systems research is
classified by Orilowski and Baroudi (1991, p.5) as positivist if only there would be quantifiable variables, hypothesis testing, etc.

- **Interpretive research** has as basis hermeneutics and phenomenology (Boland, 1985). It assumes that access to reality is only made possible through social constructions as a language and shared meanings. The phenomena is approached through the meanings that people assign to it and in Information Systems (IS) it is aimed at understanding their context but also understand if the IS influences or is influenced by the context (Walsham 1993, p.4-5). Main focus of this type of research is on the complexity of human sense (Kaplan and Maxwell, 1994).

- **Critical research** assumptions are made on the basis that social reality is historically established. People might try to change their social and economical circumstances but they are limited in doing so by social, cultural and political domination factors. Critical researchers do this by aiming at conflicts, contradictions and opposition in the contemporary society and pursue instead to be emancipatory (Myers, 1997).

### 2.3 Research Design

Research design is a widely used term in research papers. Even though the term is used in different ways throughout different works, its aim is to capture the creative work in research. To design means in this case to give shape and compose something for the purpose of the work, which previously did not exist. This needs to be done in such a way that the question at issue will be answered (Ahrne & Svensson, 2011, p.20). A research design consists of all the choices the researcher/student makes in order to give answer to the following questions:

- Which methods are the most suitable for collecting or producing data, which can be used to answer the question(s) at issue?
- Which empirical objects (Text, pictures, interview persons etc.) should be chosen for the purpose?
- How will the empirical data be analyzed and processed?
- How can we make the project trustworthy?
- Can the result be generalized and applied to other situations?
• What ethical questions are at stake (if any) and how should they be handled (ibid)?

2.4 Choice of method

I have used different resources in my work in order to gather as much theory as possible to better understand MBT and its basics. As Flyvbjerg writes (2006) theory is needed to become a beginner and experience (or field studies) is needed to move on to an expert phase (p.222). Because Model-Based Testing is relatively a new way of testing, availability of literature is limited. I will be thus limited to gather theory from some few articles in different online journals. And since these articles and books do not have an answer to my question at issue I have gathered some data from interviews from people who have been working with MBT, which will help me in answering my question at issue. This way I can move on slightly from the beginner’s phase to a more advanced phase (ibid). The data I have extracted is of qualitative character and it is needed to match it to the theoretical concepts, deepen my MBT knowledge, better understand how MBT is practiced, etc. Quantitative data would have been ideal in relating the defects to the phases but in such a short time span and from only two two-hour interviews that is unfortunately not possible in both collecting it and analyzing it. The goal in my paper is to analyze the qualitative data and try to rank the defect types as per phase, meaning which phase is most responsible for the defects and which phase is the least responsible. This makes therefore my research of qualitative character.

Questionnaires to many companies who work with MBT would have been preferable in extracting quantitative data but that is unfortunately not possible taking into consideration the timeframe for this thesis. In addition to that, they also have a few disadvantages which could give misleading answers, since one never knows who is really filling in the questionnaire; the respondent might perceive it of sensitive nature etc (Eiselen, Riëtte, Uys, JM & Potgier, 2005, p.2). Collecting data from quantitative methods would take a lot of time. It would take even more time to analyze and process the collected data, which leaves quantitative methods out of question. This makes my research of qualitative and of interpretive character.
2.5 Theoretical concepts

The theoretical concepts in this paper have supported my empirical findings and are the primary interpretation of the phenomenon I am studying. Among different uses of theory, I have used different ones.

First I have used theory to describe the phenomenon and state the importance it has as a way of AST. Second, I have used theory as a way of describing how in practice MBT distinguishes itself from AST. The last is an overlapping of the earlier two. Both theory usages have been applied to this thesis to cover MBT as a phenomenon and explain its basis. It is theory that makes it possible for us to “see something as something” (Ahrne & Svensson, 2011, p.184).

As suggested by Ahrne and Svensson (2011) and other authors, which have discussed on qualitative methods (p. 13), theory is better used for starting a research paper. Empirical findings are further used to contribute to research as non-context dependent methods (Flyvbjerg, 2006, p. 239). As Flyvbjerg writes, it is a common misunderstanding that empirical research is most useful for generating theories and hypothesis. It is quite the opposite; empirical research cannot be of value if not linked to theory. It might happen that the empirical research leads to an exception of a particular phenomenon thus generating wrong theory. What in this case is called falsification is suitable for challenging actual theories (ibid). I believe also that theoretical concepts are a necessity before moving on to empirical findings and the researcher must have a good knowledge of the phenomenon before moving on to interviews or field studies. It is difficult to assume that no theoretical knowledge is needed to write interview questions or to conduct the interview. Broad knowledge of theory will make it possible for the researcher to even interact and form a discussion with the persons being interviewed thus leading in a more effective interview, more extraction of valuable data and possibly an answer to the question at issue.

2.6 Interviews

As a secondary interpretation of the phenomenon, interviews are arguably the most common way of data collection in a qualitative study. They are a way of gathering information from someone else, information that the researcher does not know. Interviews can be categorized as structured, semi-structured or unstructured. The questions asked by the researcher can aim to cover many
questions from the question at issue or to go deep in a specific part of the topic. The alternatives depend on the question at issue and the research efforts (Ahrne & Svensson, 2011, p. 36).

In my case, interviews have been used as a method for extracting data of qualitative character. Qualitative data describes the nature, state and condition of the phenomenon (Åsberg, 2001, p. 274). During my interviews I have asked questions about MBT as a technique, the different phases and the defect distribution per phases, as well as the reasons behind that. Because my question at issue is still considered an open question and no current literature has the answer to it, I would say that interviews are a good method for data collection in my case. Interviews with people who have significant experience with MBT will be key in answering my question. Choice of who to interview, how many persons, the time and place have been factors I could not influence much on. Nonetheless I aimed for persons who have had previous experiences with MBT and actively working with it. I set the bar high at targeting some key persons at some big companies. After a lot of searching, I was able to find out that only Ericsson and Spotify are known to work with MBT here in Sweden and both companies accepted I went there for interviewing them.

A weakness of interviews is that they give a limited picture of the phenomenon, the way it is seen only by the person who is being interviewed (ibid). The researcher might receive totally different answers from different people even if he asks the same questions to them.

2.7 Case studies

Explained very simply, a case study gives the story behind the result (Yin, 2003). In other words it shows how the researcher has obtained the result and can pinpoint different phases throughout the “story”, challenging ones, problematic, not effective etc. (Neale, Thapa & Boyce, 2006, p.3). It has its own advantages and limitations. Among the advantages is worth mentioning that it provides detailed information on how the result was achieved. Another one is that it allows presentation of data from multiple methods (ibid).

Some of the limitations are that it can be lengthy; it can lack rigorosity since it is a qualitative method (a lengthy discussion regarding qualitative and quantitative methods) and it is not generalizable, meaning that different case studies can produce different results but at the same
time they have also been disposed to overgeneralization (ibid).

Leaving aside the limitations, the case study has its own process, which is similar to any research method. The steps of this plan are described as in the following:

1. **Planning** – Involves brainstorming, identifying the information needed and from whom but also that the research will follow international and national ethical research standards.

2. **Developing Instruments** – Interview protocols that guide for the implementation of the interview. This protocol should have detailed information on how the interview should be conducted, which questions shall be asked first and last, how to take notes etc.

3. **Train Data Collectors** – Only if necessary.

4. **Collection of Data** – Setting up interviews with the interview persons, explaining the purpose of the interview, why they were chosen and the expected duration. Finally, conduction the interview if the interview persons accept.

5. **Analyzing of Data** – Reviewing all documents and all data collected from the interviews

6. **Disseminating Findings** – Writing the report, revising and of course disseminating it (ibid).

My case is a typical case study since I go through all the above steps to “tell the story” behind my result.

### 2.8 Analysis method

The topic will be studied first from the gathered theoretical concepts to better understand it in the beginning. I consider myself as a beginner in this field and as Flyvbjerg (2006) also suggests, beginners and students should start from theoretical concepts to later move on to qualitative methods (p. 222). On the other hand he also writes that students learn even better through practice but not before having read current theoretical concepts of the phenomenon they are studying. The aim will be to compare theory with practice and find any possible differences but more important, the main goal is be collecting new information. The interviews will go through sorting, reducing and at last, argumentation (Ahrne & Svensson, 2011, p. 194).

**Sorting**, which solves the chaos problem will be needed to order and structure the material from the interviews. A simple interview might result in more than ten pages if written without sorting
but it is the content of the information I am out after and not the size of it (ibid). Reducing, answers the representation problem, which means that not everything from the interviews or other qualitative methods, can be part of the thesis. An interview can contain even small discussions not related to the question at issue here so cutting such parts is a necessity. Argumentation will be used at last to solve the “authority problem” which means to be heard in the research field of the phenomenon (ibid). It is of big importance that the researcher draws arguments from the collected data from the interviews and not only reviews the data. In such way, dialog and discussion will be raised with other researchers of the topic. Both epistemology and ontology will be used as methods for this research paper.

2.9 Induction and Deduction

Induction is based from data and studies from which general conclusions can be drawn. General theory or conclusions can be drawn from repeatable observations and empirical findings (Ahrne & Svensson, 2011). Deduction is the opposite of induction. By deduction one tests hypothesis drawn from theory and with the help of empirical findings either confirms or rejects the theory. A combination of both is abduction that is more suitable for case studies (ibid).

In this paper a combination of the two (abduction) is used. My two interviews will serve to draw (hopefully new) conclusions, especially since I will be asking the same questions at two different companies which seem to have different MBT approaches (more on that to come). At the same time I will test the current theory about MBT by conducting these two interviews by reaffirming the current concepts or by reaching different conclusions. Induction would have been my first choice for the purpose of this paper. It would have been idealistic to work myself with MBT at a company while I wrote this paper so that I could draw some conclusions from my empirical findings but that was not possible. By doing such an empirical study I would have been able to get a better view of MBT and better answer my question at issue.

2.10 Triangulation in a study

Triangulation means to combine different types of data, theory and different methods on which other researchers have conducted a study on, on a specific topic. This way it is easy for the
researcher to learn from these previous studies, relate to them and draw own results. By doing so, the credibility of the study raises for both the researcher itself and for the readers (Ahrne & Svensson, 2011, p.28).

Unfortunately, there is no literature that writes about my research question and no previous studies have been conducted on that which makes triangulation difficult. I will however make a comparison of the MBT approach of Spotify and Ericsson, which should be a reassuring fact for me for the theory I have gathered but also for the readers. That way I will relate to MBT from the current literature, from Ericsson’s perspective but also from the perspective of Spotify. So data triangulation is made possible from the interviews at two different companies who use MBT in two different approaches. As Guion, Diehl and McDonald have written (2002), data triangulation is arguably the most used type of triangulation as it is easy to implement (p. 1). By having this type of triangulation from two arguably top companies within their industry and from some of their employees with much experience within the field of MBT, credibility of the work is raised considerably.
3 Theoretical Framework

In order to write and discuss about MBT, one must first understand how traditional AST works so that it is easy to distinguish the two of them from one another and point out what is done differently in MBT. In this part of the chapter I will be clarifying some definitions and concepts related to AST, which are the basics of automated software testing.

3.1 Background to AST

Below are some basic definitions on Automated Software Testing, which shall serve to later describe MBT:

- **Test event** – Could be a part of a test that is a sequence of several tests.
- **Test case** – A sequence of test events that fulfills a purpose when run sequentially.
  
  An example of a test case which includes three test events, can look like the following:
  
  1) Give a variable a specified value
  2) Call a function which:
     a) enters that variable in field A and field B, and
     b) updates filed A and B in a database.
  3) Check if the fields have been updated.
- **Test suite** – The collection of different test cases.

So that the above-mentioned test can be automated run on a computer, it must be written in programming language that the computer understands. The outcome is a test script while a computer readable collection of test cases forms together a test suite. The code of the test scripts is called script code and it scans the source code when run. The source code is the test object, which can vary from a small part of the application to the whole application code (Holmberg, 2000, p.10).
Automated tests and testing are not to be mistaken with each other as they have differences aside the similarities. Here is a brief definition for each, how they differ from each other and how they are alike.

Automated test – Examine the result of one or more test suites but the execution and examination of the results is done manually. It requires a lot of manual pre- and post-processing. Under pre-processing setting up the environment takes some considerable amount of time. Under post-processing, meaning that the test has been run and collection of results is expected, the responsible persons check test logs manually. A technician usually does this after applying changes to a test subject to verify that the new changes are correct. More on this will follow later in the paper (Fewster, 1999).

Automated testing – Unlike automated tests, all job is done automatically including pre- and post-processing. It runs as a scheduled job independent of whether the test object has changed or not. It needs therefore no supervision and it is usually done during night when hardware-processing resources are vastly available (Fewster, 1999).

### 3.2 How does automated testing work?

Test scripts including test events, test cases and test suites are written in scripting language (languages supported by the OS the script will be running on). These scripts are developed...
manually and require a lot of manual effort. Updating them is also time-consuming, as the test cases need to be updated when the test objects change, thus leading to the need for easily maintainable test scripts, which I would say from my own experience is not an easy task. Parameterized scripts are preferable over hard coded ones and I assume this is a general rule in software development but that does not solve the easy maintenance problem. There is however another way of creating a script by recording manual runs of each test case. This way of creating a script is intended for black-box testing. It is optimal because opening, minimizing and doing other actions in windows is most of the time the same and they can be repeated easily but is not flexible when it comes to editing which is needed when changes occur in the test object (Holmberg, 2000, p.14).

Test review or test results can be evaluated automatically. This is categorized into dynamic and post-execution comparison of the test results with a pre-given, expected result. Dynamic comparison is made while the test is running and results are posted to a log after completion of a test case in a test suite. Post-execution comparison differs from dynamic because it runs all test cases and suites before starting the comparison (ibid). Several test scripts are usually handled by a test system, which may come bundled with the test tool or could be built by the project itself to meet up exactly the project requirements and environment (ibid).

3.3 Categories of AST

There is not one type of AST and the different kinds that exist range from testing functionality, security to performance, etc. Different projects have different goals thus making AST different from one project to another. AST is categorized into three major categories: black-box, white-box and grey-box testing (Garret, p.14, 2009).

*White-box testing* tests a system’s software. Unit testing and code coverage are two typical examples that fall in this category of testing. The main goal of this category is to check that the software’s functional requirements are fulfilled. Knowledge of the code is required (ibid). *Black-box testing* calls different functions using the user interface of the system. In this category of
testing, only the output in the user interface is tested. The system and the low level code is unknown, thus the name “black box” and that is how this category differs from white-box. Just like in real life, if you mix two crayons, black and white, the result would be grey (ibid). Grey-box testing is a combination of black- and white-box testing. Sometimes the user interface might fail to show an error because of a problem in the software code and therefore missing underlying problems in the database layer or the application logic, which are separated from the GUI. Gray-box requires knowledge of the most common parts, or prone to error parts in the application logic or database layer in order to get satisfactory results (ibid). The just mentioned categories are further categorized in the following under categories:

- **Unit testing** – tests units in software’s the source code.
- **Regression testing** – tests that previous functionality is still working even after changes in the source code.
- **Functional testing** – verification that the system meets the functional requirements.
- **Performance testing** – verification that the performance requirements are met.
- **Stress testing** – verification that system does not crash under heavy load.
- **Concurrency testing** – tests that the system can handle simultaneous users and threads.
- **Code coverage verification** – measure the percentage of code used by a test suite (Garret, 2009, p.15).

AST is best used in regression testing. Regression testing is done throughout the software lifecycle, as bugs in the real world will always exist but they are often minor and no big changes are required in the source code, however these changes made in order to correct the bugs must be tested and see if they affect the system negatively (Holmberg, 2000, p.14).

Unit testing is the lowest level of testing. A unit’s size may vary from a single function to a whole class and in the case of individual unit testing only one particular unit is tested, separated from the others.

Integration tests are part of unit tests and are used to test the integration between units and check how they cooperate with each other. The integration between units could lead to problems even though they might work fine individually (ibid).
3.4 Why Automate?

As shortly described earlier in this paper, even though programmers give the best effort in delivering error-free software, the truth is that errors will always exist. Testing before delivery of the software to the client, and even earlier stages of the software development must be done. It is costly for a company to find the errors in later stages or after delivery, so the earlier, the better and less costly the errors are. By cost it is meant the time and resources needed to fix the error, all translated into money. Manual testing will also find errors but it takes more time than AST. The process will then become monotonous to the individuals and therefore lead to unintentional miss of errors. Upon making the decision to automate, it is to be taken into consideration the number of times the test will be repeated. There is no point in automating a test if it will only be used a couple of times because the initial setup of automating a test is high. It takes a considerable amount of time to configure the system and even more time writing the script. To make the decision if the test will be automated, it is of value to compare the total cost of automation with the total cost of manual testing. The responsible persons should not be set back by the initial, high cost automating the process (Holmberg, 2000, p.15).

The table at the beginning of the next page shows the error cost through the software’s development cycle (Rashka, 1999, p.8).
<table>
<thead>
<tr>
<th>Phase</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>$1</td>
</tr>
<tr>
<td>High-Level Design</td>
<td>$2</td>
</tr>
<tr>
<td>Low-Level Design</td>
<td>$5</td>
</tr>
<tr>
<td>Code</td>
<td>$10</td>
</tr>
<tr>
<td>Unit Test</td>
<td>$15</td>
</tr>
<tr>
<td>Integration Test</td>
<td>$22</td>
</tr>
<tr>
<td>System Test</td>
<td>$50</td>
</tr>
<tr>
<td>Post-Delivery</td>
<td>$100+</td>
</tr>
</tbody>
</table>

*Table 1, Error Removal Cost over Different Phases in Software Development (ibid).*

Quality is another positive aspect in automating a test. If manual testing takes a long time and specific tests are hours long and are to be repeated daily, it can lead that the testers get bored and tired and therefore miss errors. Automated testing needs no supervision and will carry out the tests correctly and up to specifications no matter the duration and the frequency (Holmberg, p.17, 2000).

### 3.5 Phases of AST

This section is very important to describe the different phases of AST, which is strongly connected to my question at issue. I will focus on the different stages that test scripts go through, before and after they are created, let apart tool acquisition and other details. These phases are:

- Test planning
- Test design
- Test development
- Test execution
- Maintenance

The first stage, test planning, identifies standards and guidelines required to create a test environment such as: hardware, software and network requirements. During this phase, the testing team also specifies other requirements such as test schedule, defect tracking procedures and the tools needed for it (Rashka, 1999, p.13). Test design, defines test conditions, standards and the ways testing will be performed. The design is an outline of the framework including the boundaries and the scope of the test program (ibid). Test development is the phase under which the test team develops the test scripts following the guidelines, the conditions and the standards that were specified during test design (ibid). Test execution phase is where the developed scripts are run to test the desired parts of the software. The outcome of the results is also analyzed at this stage (ibid). Maintenance is a form of test script update where the scripts are updated if the requirements or the software change. It is a form of test development (ibid).

From my own experience and other senior testers, the above categories can be put together as 1) Test analysis, 2) Test design and 3) Test execution. The defect distribution per phase will be related to these three phases which all require human effort except perhaps test execution which can be automated. Less manual effort is required in a MBT approach. The following chapters will cover this. Model-Based Testing is a black-box software testing method seen as a complementary approach for solving problems with test script maintenance in traditional AST. It is both time consuming and costly to update the test scripts manually every time the software or the requirements of the software change (Merilianna, Puolitaival & Pärssinen, 2010). MBT makes generating new test scripts easier.

### 3.5.1 MBT Phases

The process of MBT can be divided into three phases: 1) Modeling, 2) Test generation and 3) Test execution.

**Modeling**: The basis for MBT is the functional requirements of the system. These requirements are what lie underneath the GUI of the system, within the application logic. A test model is
created once the requirements are set. The models are created based on the different requirements and the different types of combinations that the designers can create so that they represent the intended behavior of the system under test (SUT) (Özay, 2007). Both knowledge of the input and knowledge of the output are required for the model. Input data is necessary for test execution while output data is used for validating the tests through comparison with expected results. *That said, a model describes how a system should behave in response to an action* (ibid). Modeling can be done in different modeling languages such as: UML, Markov chains, state transition diagrams etc. The modeling represents either the intended behaviour of the SUT or an expected behaviour of the environment of the SUT. The later one will restrict the achievable inputs to the model and hence it also acts as a test selection criterion. Random environment models that are defined by the user describe a typical pattern of **incentives** to the SUT (Utting, Pretschner & Legeard, 2006).

![Figure 2, Models of the SUT and its environments](image)

The picture above shows the possible combinations of the environments of the SUT and the models of the SUT. The vertical axis represents the SUT and the behaviour, which are both modelled there while the horizontal axis represents the environment, and how much of it is modelled (ibid). Point S in the above diagram shows a model that has knowledge of all the details of the SUT but knows nothing about the environment. Point E, which lies in the horizontal axis, has knowledge only about the environment but knows nothing about the anticipated SUT behaviour. SE point in the grey-shaded area is the perfect combination but unable or at least not practical to achieve in real life because of the complexity of created model as it would include a lot of detail thus resulting in a model as complex as the SUT itself. In real life, M1-M3 are most
likely to be achieved since some level of abstraction is always needed. Abstraction can be performed in two ways: it can either be induced by the modeling language itself or by the modeller who neglects certain information. Both of this methods are used but the second method can be further divided into: Function abstraction, data abstraction, communication abstraction and abstraction from quality-of-service (ibid). Function abstraction excludes some of the functionality of the SUT. This is a practice much used in MBT. The team judges that certain functionality is not critical for the project’s goals so there is no need for such functionality to be tested (ibid). Data abstraction is a form of abstraction for input and output. During input abstraction, parts of the inputs of the SUT are left out. Output abstraction simplifies the outputs of the SUT. Communication abstraction is mostly used in protocol testing (ibid). Abstraction from quality-of-service is used for timing, security, and memory consumption. In the case of timing abstraction, the logical time is taken into consideration over the actual physical time (ibid).

**Test generation:** Is done automatically through different algorithms by the tool once the models are created. We can distinguish between the following algorithms:

- The test design algorithms are *requirement-based* which strive to cover all the specified requirements.
- Through *coverage criteria algorithms*, the test suites are generated with the aim of covering the desired degree of the model.
- *Walking algorithms* are the ones that generate test suites based on a specific sequence (*Puolitaival, 2008*).
Test execution is categorized in online and offline modes. Online testing is made during test case generation. The tool not only generates new test cases but it also executes the ones that are ready. Offline testing execution comes after the collection of the generated tests. Offline testing requires heavy algorithms and therefore is done after test case generation. Online testing uses less heavy algorithms and can be run while new test cases are generated (Hartman, 2006).

In the below picture we can see the testing flow and difference between the two modes. The main difference is that while choosing and performing offline testing, we evaluate and report the results as part of the process without necessarily having to be satisfied with the results. In online testing, the process is repeated continuously until the team is the objectives are met.

The testing tool chosen by the team is much affected by the test execution strategy. For example,

![Figure 3, Online vs Offline testing (Hartman, 2006, p.206)](image-url)

in the online case the test generation tool is connected to the SUT where it continuously translates inputs and outputs between the test automation system and the SUT. In the offline testing execution strategy, the test cases are created first but unlike the online testing; the translation of the inputs is done before they are executed on the SUT (Hartman, 2006, p.206).

The picture below shows an overview of how MBT works without taking into account the above strategies.
3.5.2 Finite state machines and infinite state machines

MBT can be used to test both finite and infinite state machines. The finite state machine has a specific number of states it can be in which makes test coverage a bit easier than infinite state machines where full test coverage can never be achieved because the number of test cases that can be generated is infinite (Özay, 2007). Below is a simple example of a finite state machine to better explain how MBT works in that case. Before I do that, it is important to clarify that a finite state machine consists of the quintuple I, S, T, F, L where:

I – are the inputs of the system
S – are the states of the system
T – are the functions that determine of a transition will occur if an input is applied to the system under a specific state
F- are the final states the system can be in and
L – is the state in which the software is lunched (ibid).

Now moving on to the example. Let us take a simple a somewhat simple light switch. The lights can be turned on or off with one input – the main switch. The intensity of the light has two switches, which dim the lights or increase their intensity. The intensity has three levels: dim,
normal and bright. Assuming we start the simulator with the lights off, which is the initial state of the machine, we turn the light on and the intensity is normal by default. The quintuple in this case is:

\[ I = \{ \text{<turn on>}, \text{<turn off>}, \text{<increase intensity>}, \text{<decrease intensity>} \} \]
\[ S = \{ \text{[off]}, \text{[dim]}, \text{[normal]}, \text{[bright]} \} \]
\[ T: \text{<turn on>} \text{ changes [off]} \text{ to [normal]} \text{ <turn off>} \text{ changes any of [dim], [normal], or [bright] to [off]} \text{ <increase intensity>} \text{ changes [dim] to [normal], or [normal] to [bright]} \text{ <decrease intensity>} \text{ changes [bright] to [normal], or [normal] to [dim]} \]
The inputs do not affect the state of the system under any condition not described above
\[ F = \text{[off]} \]
\[ L = \text{[off]} \]
The SUT in the above example has a finite number of states thus making it easy for a MBT tool to generate all the test cases and it makes it easy for the testers to run the generated test cases. The above can be modeled in the following state transition diagram:

*Figure 4, State Transition Diagram, Light Switch (Özay, 2007)*
The diagram (figure 4) is then read by the MBT tool to later generate the test cases needed to simulate the model. Different tools work with different modeling languages and generate test cases in different programming languages.

*Infinite state machines* can generate an infinite number of test cases but there is simply no time. Under these conditions it is more important to test the behavior of the system rather than the test coverage of the system. That leads to a selection of a finite number of test cases. This selection is made by applying different algorithms or by choosing randomly some test cases. No matter the strategy selection, it should focus on maximizing error detection and the cost of executing the test suite (Beaulah Vineela, 2009, p.7).
4 Empirical findings

My empirical findings are based on interviews at two companies in Sweden that use MBT as their way of testing their products. The companies where I conducted the interviews are Ericsson and Spotify, both internationally well known for the services and products they offer.

My first interview at Ericsson took place in Kista at Ericsson’s offices there. I interviewed two software developers. Their names are Håkan Fredriksson and Ebrahim Amirkhani. Both have been working with MBT since 2007. The interview at Ericsson was of a qualitative nature where questions were asked and a discussion was formed around them. The qualitative data I have extracted from this interview come as both my understandings from the answers I received and as direct answers to my questions from the respondents.

My interview at Spotify was with Kristian Karl who is currently a test manager at the company and at the same time founder of GraphWalker, an open-source MBT tool. The interview took place online as a video call and it was of qualitative character. Again, as with the interview at Ericsson, the data I have collected from this interview is my viewpoint from the answers to my questions but also straight answers to my questions.

4.1 Ericsson

Ericsson is a global company that focuses on producing network solutions for different means of communication. For the moment, Ericsson is the world’s biggest supplier of mobile networks chosen by about “half of the world’s operators with commercial mobile broadband networks” (Ericsson, 2013). With LTE being the latest technology, the researchers at Ericsson are still continuing their work on Global System for Mobile Communication (GSM), Wideband Code Division Multiple Access (WCDMA) and Code Division Multiple Access (CDMA). Other than mobile networks, the company is a big player in the market of core networks, microwave transport, IP networks and fixed-access solutions for copper and fiber. All network products account for 55 percent of Ericsson’s net sales (ibid). The rest of their sales come from services. They focus on delivering professional services for the Information and Communication
Technology (ICT) sector in areas such as: consulting, system integration, network rollout and customer support (ibid).

4.1.1 Interview Questions

During the interview at Ericsson with Fredrik Fredriksson and Ebrahim Amirkhani, I asked them general questions about MBT and specific questions related to my question at issue. The following are the questions I asked:

1. Why MBT?
2. How much and what is tested with MBT?
3. MBT better suited for functionality, security, performance or usability?
4. Which tools are used and what modeling languages?
5. How is a MBT model built? Are there difficulties in this phase? What do the difficulties depend on?
6. Can MBT discover defects that the traditional testing has not discovered?
7. Are there defects after implementation that MBT has not discovered?
8. If so, which MBT phase is responsible for not discovering these defects? Why?

A summarized answer for all the questions above will follow in the next two sections of this chapter.

4.1.2 MBT at Ericsson

Ericsson has been working with MBT for some years now. All started from a conference that Conformiq (company that develops MBT solutions) was holding where they showed their MBT tools to the attendees. Doubtful but still curious, the testing team at Ericsson gave it a try. The results from the tryout were much better than the traditional way of automated testing. The time effort was minimal so they decided to stick with it. What drove them into MBT were the time consuming and the expensive maintenance of the test scripts under the design phase. From the very beginning, the goal at Ericsson has been to automate everything but since MBT is better suited for black-box testing; the main categories of testing carried out with MBT are regression,
functional and even some bits of non-functional testing such as system performance and stability. It is of best profit if non-functional testing is done parallel with functional testing.

4.1.3 The MBT Approach

What started from curiosity and an invite to a conference from Conformiq, MBT has been present at Ericsson for many years now and it is the kind of testing that covers almost all of functional testing. Other than that, even regression and integration testing is done with MBT. It is though not suited best for performance testing or other kind of testing which is not repeatable. The reason for that is that it is expensive to automate everything if the scripts will be used once or a couple of times. Exploratory testing is better suited for that purpose. MBT at Ericsson consists of three major phases: Test analysis, test design and test execution. After the requirements are set, the team goes on to test analysis and feasibility analysis. Through feasibility analysis the project team analyzes whether the desired tests can be implemented. The analysis is mainly based on the following factors:

- Time – Is there enough time to implement the planned testing? Is there a deadline for the project and how strict is that?
- Financial – Does the planned testing lie within the budget or will it require extra financial efforts? Are extra financial inputs allowed?
- Legal – Does the testing project break any laws in any way? If so is there a way to adapt to the laws?
- Technical – How does the current hardware and software meet the needs of the project? (Amirkhani & Fredriksson, personal communication, March 13, 2013)

Under test design, models for the MBT tool Conformiq are designed in state diagram Unified Modeling Language (UML). Once the models are created, they serve as an input for the MBT tool to generate the test scripts, which are also considered as part of the test design phase. Test execution is done offline, as the algorithms that generate the script are heavy to be run at the same time as new ones are generated. Online testing is not done at Ericsson. Because their Systems Under Test (SUT) are infinite state machines, they go through a selection strategy for the test scripts. Without doing so, an enormous number of test cases would be generated which
would be also very costly. Functionality is the main factor in the strategy selection as the team is most interested in testing the functional requirements of the product. The role of MBT is crucial in the modeling phase at Ericsson. The generated scripts/suite are then run, as they would be in the traditional way of testing.

The figure above is an overview of the MBT approach at Ericsson. First, the requirements are set, and then the system models are designed based on the requirements. The system models serve as an input for the Conformiq tool that generates test scripts grouped in different test suites. When regarding the errors that show up after implementation, Ericsson had no statistical data about which phase of testing these errors were related to. Interesting was to know that the number of errors has decreased drastically in comparison to the traditional way of testing. From their own experience, Fredriksson and
Amirkhani (personal communication, March 13, 2013) acknowledged that none of the errors they encountered after implementation were related to the MBT tool, nor were they related to the execution phase. Test analysis and the modeling phase were seen as the most problematic where 100 percent of human effort is required. Errors were sometimes found during the execution phase but that does not mean that that phase is the problematic one as the generated test scripts serve as the input for it. The generation phase in itself has as input the models designed in the modeling phase thus pointing to that phase as the problematic one. When such an error is discovered and depending by the severity of the error, the test team can either decide to update the test script/suite manually or redesign the model to recreate the script/suite from the MBT tool. The team practices both ways but the first one is practiced for smaller kind of errors though the team has no model this way and no such can be used later in need of updating it. After saving the created models and after the testing has moved to a mature state in both test coverage and quality, the project moves on to another phase which is called “design follow-up” or simple maintenance. Errors can albeit show up even in this phase and in this case it is another team who takes care of error correction. When asked why the test analysis phase is also seen as problematic and if it was possible to avoid those errors, Fredriksson and Amirkhani (personal communication, March 13, 2013) answered that those errors can be avoided. It is not something that the team cannot do, does not have the competence or other technical issues. It is rather the short deadlines the project has which make the team rush during the way and therefore missing some easier errors.
4.2 Spotify

Spotify is a digital music-streaming service that gives its users on-demand access to millions of songs on different devices and platforms. With social medias growing rapidly in the last years, Spotify is also part of it through letting its users share songs and playlists with each other by integrating its services with other social medias such as Facebook. Artists on the other hand can work with Spotify and publish their songs there (Spotify, 2013).

4.2.1 MBT at Spotify

I could not help but notice that Spotify had a different MBT approach if compared to Ericsson even though their software is also infinite state software. Same questions that were asked at Ericsson were also asked at Spotify.

As an agile company that works in 3 week long sprints, Spotify focuses more on new functionality. Added functionality to the software might however affect older code thus making regression testing a must. Spotify uses Graph Markup Language (GraphML) as a modeling language which is easier to use than UML but does not provide all the functionality of UML. As opposed to UML that provides a lot of modeling possibilities and requires deep knowledge (there are even jobs as UML designer!) of the language, GraphML is very simple and non-testers can get comfortable with it quite quickly (Karl, 2013). The models are of course the abstraction layer in the automation process. The tool they use for generating the test scripts is called GraphWalker. It generates new test scripts and executes the ones generated at the same time, in other words online testing. Offline testing is also done. Since their software is infinite state, the generated scripts must go through selection. Spotify does the selection randomly and executes these random test scripts until they are satisfied with test cases they have. So MBT at Spotify is used iteratively through all the phases, unlike at Ericsson. Below is a simple graphML example to show how different it is from UML.
The test scripts are later implemented in Jenkins where track of errors per unit is made easy. Spotify did not have any statistical data about the relation between MBT phases and errors. They did however state that most of the bugs are fixed within the sprint so they never go further. These bugs are informally reported and are treated as in-sprint bugs making it difficult to generate statistical data for them. These errors show during test execution phase. Interesting was also that Spotify looks after UI testing. The tool that works with the modeled state charts is called Sikuli. After all, Spotify’s customers interact with the software strictly through the UI so black-box testing is also needed. The UI testing with MBT is not run separately but alongside with white-box as a complement as the picture below shows (ibid).

Figure 6, graphML example (Höglund, 2013).
As seen in the picture, the Test Result Service (TRS) shows the working and non-working features together with an attached screenshot for each feature. When asked about which phase is considered the weakest in the MBT phases, Karl was unable to give statistical figures. But from his own experience, test design was most probably the weakest phase. Test generation was also a problem when regarding to the selection strategy but the tool itself is able to produce infinite amounts of test cases, which is not directly a weak point of the phase itself but more of a problem in test execution where selection of the test cases to be run is made.

**Figure 7, TRS at Spotify (Höglund, 2013)**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Result</th>
<th>Model</th>
<th>Edge/vertex</th>
<th>Attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>e_Init</td>
<td>1361030163722.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>v_NuxLoginPrompted</td>
<td>1361030166862.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>e_NuxLoginHere</td>
<td>1361030170020.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>v_LoginPrompted</td>
<td>1361030174654.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>e_LoginSpotify</td>
<td>1361030178768.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>v_LoginSpotifyCreds</td>
<td>1361030183263.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>e_ValidSpotifyId</td>
<td>1361030190436.jpg</td>
</tr>
<tr>
<td>Login</td>
<td>OK</td>
<td>Login</td>
<td>v_InitialView</td>
<td>1361030196134.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>e_Init</td>
<td>1361030199270.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_InitialView</td>
<td>1361030202387.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>e_GoToSettings</td>
<td>1361030207543.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_Settings</td>
<td>1361030215239.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>e_DisablePrivateSession</td>
<td>1361030218860.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_PrivateSessionDisabled</td>
<td>1361030225255.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_ShowOnFEnabled</td>
<td>1361030229882.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>e_EnablePrivateSession</td>
<td>1361030251689.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_PrivateSessionEnabled</td>
<td>1361030256320.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_PlayTrack</td>
<td>1361030271634.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>OK</td>
<td>FacebookScroblng</td>
<td>v_Playing</td>
<td>1361030281173.jpg</td>
</tr>
<tr>
<td>FacebookScroblng</td>
<td>FAIL</td>
<td>FacebookScroblng</td>
<td>v_NotScroblng</td>
<td>1361030285690.jpg</td>
</tr>
</tbody>
</table>
5 Analysis and discussion

5.1 Automated Software Testing and Use of Model-Based Testing

In this chapter, the results of the case study are analyzed.

MBT at Spotify and Ericsson is done as a better alternative when compared to traditional AST. The main advantage of MBT is test script generation, which in AST takes a lot of time not only in generating them for the first time but also updating and maintaining them. That is further translated into costs for the company (Rashka, 1999, p.8). Regression testing is always needed when the application to be tested is changed, for example when new functionality is added or bug correction is done (Holmberg, 2000, p.14). Upon adding new functionality to the when carrying out bug fixing other units of the application must still work as they should (Holmberg, 2000, p.15). Test coverage is done better with MBT when compared to AST. MBT can reach a much higher testing coverage with much less effort (Binder, 2011, p. 31). This can be especially problematic in the case of infinite state machines or more frankly speaking: impossible (Özay, 2007).

Main use of MBT is white-box testing where the UI is never involved with the test scripts generated. The reason for this is the functional requirements, which are set even before the test analysis phase. These requirements are too specific and require deep knowledge of the test code, things which are not accomplished by running tests through the user interface. The functional requirements are part of functional testing through whom the test team assures the system meets the functional requirements and as Utting, Pretschner and Legeard (2006) state, the functional requirements are the basis of MBT (p.2). However that is not the case with Spotify, which uses MBT also for black box testing alongside with white-box. Even some bits of non-functional testing are done with MBT. Not only, MBT in practice stretches all the way to performance testing as Ericsson does. That is also a possibility with MBT but Utting, Pretschner and Legeard (2006) write that it is currently out of the mainstream trend for MBT (p.2).

Unit and integration testing is also used with MBT but the testing category where MBT excels and where a lot of profit comes from, is regression testing. In AST, scripts are manually updated or new ones or created each time the software is changed or new functionality is added.
Instead of manually updating the scripts or creating new ones after functional requirements change, MBT generates these new/updated scripts automatically from the models depending on the criteria or the algorithms chosen. The team might strive for functional, coverage or walking algorithms (Puolitaival, 2008). At Ericsson the team focuses on functional algorithms (H. Fredriksson, personal communication, August 27, 2013). In the case of updating the test scripts, the test team only needs to do some minor tweaks to the model in the modeling phase for later reuse of the model. At Ericsson this practice is sometimes not followed in case the test scripts generated by the models contain small errors. Those errors are fixed manually without changing the models. Not the best theoretical way to do it, but sometimes practical if judged that the model would not be used later (ibid).

5.2 Flaws of MBT

Without a doubt, MBT is more time efficient (ibid) than traditional AST and relatively less costly (Puolitaival, 2008). MBT faces yet other problems compared to AST. In infinite state machines/software, the MBT tools can generate infinite number of test scripts. Logically, the team cannot wait endlessly and are therefore forced to apply selection strategies on the desired number of the scripts to be generated (Utting, Pretschner & Legeard, 2006, p.8).

Ericsson uses no such selection strategy with the help of MBT since MBT is used only for the modeling phase (Amirkhani & Fredriksson, personal communication, March 13, 2013). Spotify seems to apply a more random selection strategy and proceed with online testing until they are satisfied with the test coverage and functional coverage (K. Karl, personal communication, April 18, 2013). Before even the selection strategy, there is also the abstraction layer in the modeling phase, which further reduces what is to be tested in the SUT. Too much detail results in a very complex model, more complex than the SUT itself. Negotiation of the abstraction level is always a must and the right balance must be found (ibid). Ericsson strives for example to cover as many functional requirements as possible (Amirkhani & Fredriksson, personal communication, March 13, 2013).

In-sprint and out-of-sprint (widely used concepts in agile methodologies) bugs are a different problem related to the question at issue. Ericsson does not make such differentiation for the
simple fact that the product of the team is later delivered to another team for maintenance. During maintenance, another team can correct the errors and that makes it an in-sprint error correction for the maintenance team but an out-of-sprint error correction for the modeling team (ibid). Spotify encounters most of its errors during sprints and these errors are fixed within the sprint without being documented formally and therefore the missing statistical data about the errors. To my opinion, both in-sprint and out-of-sprint would have helped in better answering the question. In-sprint errors occur before delivery of the product to the customer but they would have revealed further the weak points of the MBT.

My wonders so far are the time I had which was strictly connected to the choice of method for this paper. Missing literature (a known disadvantage from the very beginning), left out deduction as a method. My two interviews led to some kind of induction. Field studies would have been the most preferable for me in this case where I would get to practice at least two different tools in two different projects. I am since required to draw some general conclusions from the interviews and from the literature I found.
6 Conclusion

The goal with this paper has been to find the weak phases of MBT that are responsible for the errors that show after the product has been implemented. Once again, the question at issue to be answered is the following:

- Taking into consideration MBT phases, what is the relation between defects, their causes and their distribution per phase?

After the interviews at Spotify and Ericsson, which further confirmed the current literature, MBT is a far better way of automating tests than AST. Key for that, where MBT outperforms AST is the model creation in the modeling phase. The models then generate the test scripts and the test suites that serve as input for the execution phase. When the test scripts need to be updated, only a slight change in the model is needed which will trigger the creation of many new test scripts. That leads to the fact that further automation in the automation process is more effective than creating the scripts manually.

MBT is though not error-free and defects make it even after software delivery to the customer without being detected. Regardless of the way it is used (the cases with Spotify and Ericsson), in infinite state machines MBT usually has one weak phase, which is the modeling phase. Because of the infinite states, an abstraction layer is needed in creating the models. The team does not want to create a model that is more complex than the SUT itself but neither do they want to leave out key functionality. It seems that when making this decision errors occur and they are translated in a weak model that outputs problematic test scripts. We have to then differentiate in detail the way MBT is used at Spotify from the way that Ericsson uses MBT.

- Ericsson uses MBT simply for the creation of the test scripts, which that after the models have been created; they are sent to the MBT tool in order to generate the test scripts. The tool is later not involved in execution, which is done offline. This fact excludes automatically the execution phase from responsibilities of errors. Out of the two critical phases that remain, modeling was the problematic one because of the abstraction layer that the team must chose to avoid overly complicated models. Test generation was not a
problem since the algorithm chosen is functional and that does not lead to an endless number of test scripts. The MBT tool was never found to be problematic.

- Spotify, unlike Ericsson, uses MBT all the way to test execution. In the case of Spotify where MBT is used even in the execution phase and because of the enormous number of the test scripts generated, errors can be made in the test execution phase in selecting the scripts generated. The selection strategy for the generated scripts is a possible problem under the execution phase, especially for in-sprint errors but accounts also for some of the out-of-sprint errors. Out of the many generated scripts a selection strategy must be chosen to minimize the load on the testing tool.

The weak points in MBT at both companies show that the manual bits of MBT are the most problematic ones leaving out the MBT tool (test generation phase) as a problem. Furthermore, it showed that the test analysis, which takes place at the beginning of each project, is responsible for some defects and that is because of missing requirements or in defining the specifications for the project. Since both companies have different approaches, in Ericsson’s case the execution phase is not taken into account at all as part of MBT and therefore left out of the phases to be analyzed. Spotify uses online testing and the execution phase is also there as part of MBT. The abstraction layer during the modeling phase can leave out some good test cases.

To sum it up, I would say MBT is a far more effective way of automating tests than AST. I cannot however say which phase accounts for more errors than the other (since neither Ericsson or Spotify had such data) but the weak points of MBT are the non automated parts of modeling and execution phases, choosing the right abstraction layer and the right selection strategy respectively.
7 Personal Reflections and Future Research

Answering my question at issue was not an easy process. Since MBT is a newer way of testing, not many companies use it (at least here in Sweden) and that made it even more difficult to conduct more interviews.

The most difficult part was getting a direct answer to my question from both Ericsson and Spotify because the question was also new to them and it was hard to relate to the question with their way of working. None of the companies have statistical data about the defect types per phase. The interviews were however the most giving part of my research as I could understand how companies put MBT into practice (and in two different approaches). Even if it is clear that it is a better way to automate test compared to AST, many teams stay away from it. This is the case with Ericsson where in Linköping they work with AST but the team in Kista has been using it actively for many years.

Interesting enough is also how different approaches of MBT can lead to different conclusions. As described in the conclusion, Spotify uses MBT all the way to test execution since most of their testing is done online. Ericsson on the other hand uses MBT only for the creation of the test scripts and then runs the scripts offline. That is partly because of the heavy algorithms that are used for the creation of them but also because they strive for functional algorithms when creating the test cases. Hence the test execution phase cannot be accounted for any errors because it is done like in the traditional AST. Sure enough errors might show during that phase but it is not because of the MBT approach. That could be the reason why Ericsson struggles mostly with the modeling phase and somewhat the test analysis phase. Spotify on the contrary, finds errors even in the test execution phase because of their approach. The consequences for each company are clear based on the different approaches they use. They differ especially in the test execution phase. Spotify faces the problem of the test selection strategy since the use MBT all the way. The endless number of scripts which can be generated undergo a selection. In the worst case, good scripts might be left out. They also choose another way to run MBT which is a very random execution ‘until they are satisfied’ with the test coverage achieved. This can also lead to missing important test cases even though it does not happen in their practice.
A common potential weak point they share is the modeling phase. Both companies must use some kind of abstraction layer, which can again leave out potential good cases.

As a rather new way of testing, many open questions remain in this area leading to future research. However during my research I noticed the difference between the in-sprint and out-of-sprint defects. Gathering such information, targeting the problematic phases and solving the issues would improve both inner and outer effectiveness of the company. That led me wondering about the following questions, which could be a good start point for future research:

- What is responsible about MBT defects that appear during in-sprint bugs?

Another question that still remains open is:

- What are the origins of the defects (Limitation of the model? Testing criteria? Tool? Testing process?)

All the above questions are challenging and if I had the possibility to work with MBT at a company I would take the chance and investigate further into them.
Reference list

  ISBN: 91-7871-969-0
- Fredriksson Häkan (2013, August 27), E-mail interview.

- Utting Mark, Pretschner Alexander, Legeard Bruno (2006), A TAXONOMY OF MODEL-BASED TESTING, Department of Computer Science, The University of Waikato.
- Vineela Pasupulati, Beaulah (2009), Traceability in Model Based Testing, 5-7, Master Thesis. Mälardalen University, Sweden.


**Interviews**

- Fredriksson H. (2013, August 27). E-mail interview.