Assessing Model-Based Testing: Results from an Industrial Case Study
Christoph Schulze, Dharmalingam Ganesan, Mikael Lindvall, Rance Cleaveland
Fraunhofer Center for Experimental Software Engineering,
{cschulze, dganesan, mikli, rcleaveland}@fc-md.umd.edu
Daniel Goldman
Global Net Services Inc.
dgoldman@gnsi.com

ABSTRACT
Companies continuously look for ways to improve their processes and cut costs using new technologies. Model-based testing (MBT) is a technology that has the potential to be such a cost saver.

In this paper, we present a case study in which we compare manual testing processes with MBT processes based on finite state machine (FSM) models.

The system under test (SUT) is a professionally developed web-based data collection system that is now in use, allowing researchers to exchange findings of laboratory analyses regarding food borne illnesses. Two versions of the SUT were tested to evaluate the regression testing abilities of the two approaches.

Manual and model based testing were carried out by independent teams that had access to the same resources but did not interact with each other.

We compare the effectiveness (issues found) and efficiency (effort spent) of the two approaches. The results show, for example, that the manual testing process observed in this study required less preparation time but its coverage was somewhat uneven, while the observed MBT study required more preparation time but was more systematic and detected more issues. Furthermore, the two testing approaches detected different types of issues.

We also discuss lessons learned, how each technology can be improved, as well as how the best parts of each technology can be used in a hybrid approach to improve the testing processes.

1. INTRODUCTION
Software testing is a vital, yet expensive part of the software development process and companies are continuously searching for ways to optimize their workflows to cut costs and to increase quality. A survey of software developers and testers found that the need for software testing is increasing as businesses face pressure to develop more sophisticated applications in shorter timeframes.

Organizations are looking for ways to eliminate the risk of launching poorly-tested applications. Even though most organizations report that they spend up to 50% of their total software development costs on testing, yet over half of the IT professionals surveyed said their organizations did not spend enough on testing. A study at the University of Cambridge has estimated the global economic cost of software debugging with $312 billion per year.

Automated software testing is an umbrella term for different techniques that automate one or several parts of the software testing process. Automatic execution of (handwritten) test cases has been widely adopted in the industry already. The automatic generation of test cases, from source code or via models has only found acceptance in industries with high quality requirements so far however.

Model based testing (MBT) is a testing technique where models of the system under test (SUT) are used to automatically derive test cases. There are a variety of different modeling techniques for MBT: a survey of them can be found in [1]. The MBT-variant used in this study uses finite state machines (FSMs) because the modeling process of this technique is intuitive and is therefore more accessible. Other modeling techniques such as extended finite state machines and (EFSM) and abstract finite state machines (AFSM) are not as intuitive and require software coding knowledge.

The benefits that a new technology brings are not the same for each end-user. Often the applicability of a technology depends on the domain that it is used in. In order for developers to adopt new technologies they need proof that the infusion of said technique offers the promised benefits for their organizational domain [2]. So far MBT has only been analyzed for its effectiveness to find bugs but not for its efficiency at which it does it. Also both the effectiveness and efficiency have not been compared to other approaches yet in a study with real testers.

This paper presents a case study were we compared the manual testing process of our partner company GlobalNet Services, Inc (GNSI) with automated testing using a FSM-based MBT approach. The goal of the study was to investigate if, or under what circumstances MBT performs differently than the existing manual technique. For this we collected data about the effectiveness (detected issues) and efficiency (effort) of the two approaches. A test engineer at GNSI conducted the manual testing and the MBT was conducted by a Fraunhofer scientist. The two testers worked independently of each other but had access to the same resources (e.g. documentation, developers).

It was a unique opportunity to study testing in a real life scenario using a real system together with an industry partner. These results say more about the performance of experts in real situations over weeks as compared to students conducting experimental tests for a couple of hours. Since it is common that the same system has to be tested several times, both techniques were applied on different versions of the SUT to examine their characteristics over time.

2. Related Works
Model based testing has a lot of potential but its effectiveness and efficiency still needs to be analyzed further so that potential users have data to use in their decision making process. So far most publications focused on the number and the type of the issues that were found and not on the effort to apply the approach [3][4][5][6].

Kervin et al [5] and Vieira et al [6] apply MBT to systems with a graphical user interface (GUI). The focus however is mostly on

2 https://softwaretechnews.thedacs.com/stn_view.php?stn_id=43&article_id=90
3 http://www.prweb.com/releases/2013/1/prweb10298185.htm
presenting the approach and on showing that they found issues with them. No comparison to other approaches is done in their case studies.

Apfelbaum and Doyle [7] presented a case study of a real software system where they compared MBT with manually created test cases. The focus was mainly put on the modeling process and not on a deeper analysis of the effort or the issues.

Utting and Legeard [4] present an effort comparison of Manual Testing, Capture Replay Testing, Testing via Scripts, Keyword Testing and MBT. However, the evaluation is not based on real world data but on models. The models are based on reasonable assumptions but they are no replacement for real world data. In contrast to this case study collects real world test data and compares the effort and effectiveness of manual testing and MBT based on an actual system that has been developed for a customer of our partner company.

Finite state machine based MBT techniques are very mature and their models are easy to comprehend, even for people without a computer science background. There are other techniques that use different kinds of models to describe the SUT. Neto et al [1] presented a survey that lists the relevant techniques and tools. There are two general classes of MBT tools, graphical and abstract. In the first class the user creates the graphical model himself and then uses this model to test the system. The workflow in the second class is slightly different. Here the user does not draw the diagram by hand, instead he uses a formal language to describe the system. A tool generates the concrete model from this description. One example of this is the abstract state machine language presented in [8]. The abstract system description is turned into a finite state machine that can be used to test the system just like in the graphical approaches. The languages for describing the AFSM are very powerful and descriptive but they are not as easy to learn since they require programming experience. They might be a good fit for users that already use MBT as part of their V&V efforts and want to use more advanced modeling techniques.

Other automation techniques besides MBT have been evaluated. Ciupa et al [9] analyze the behavior of random testing in comparison to manual testing. However, the manual approach was only applied on three classes of a software library whereas the random approach was evaluated on more than just these three classes. Aside from that they propose a classification schema for the issues that they encountered in their case study. The case study that we proposed evaluates a real life software system and both approaches are evaluated on the same functionality of the system. We also established a classification schema for the issues that we collected. Since the system in their case study (API of a class) is very different from the system that we tested (Web based system) the two classification schemas are not comparable.

3. Background

This paper is the result of an ongoing collaboration, between GNSI and Fraunhofer, regarding evaluation and analysis of how new technologies could help GNSI improve their software development and testing processes.

As part of this collaboration, Fraunhofer presented results from MBT conducted on NASA systems. The results from the NASA studies were promising; however it was still unclear whether such an approach would work well in GNSI's environment. To address this question, in 2010 we conducted a pilot study in which GNSI provided a set of use cases for a web-based system that was already in production. Fraunhofer's task was to apply MBT to determine whether it was possible to detect issues using reasonable effort. The pilot study showed that MBT did indeed detect issues with reasonable effort. However, due to lack of data it was not possible to compare the results (effort and issues) to manual testing. To do a more detailed study of the commonalities and differences between MBT and manual testing we decided to run this study and carefully collect data to allow for comparison and analysis.

4. Case Study

As discussed above, MBT has shown promising results for detecting issues in software systems, but there is a lack of data that clearly shows the advantages and disadvantages of MBT in comparison to other testing techniques. Before introducing new technologies users want to see proof of its effectiveness prior to committing valuable resources to it. They want to be assured that the introduction costs (E.g. training, tools, hardware) that a technology requires will eventually pay off for them in the long run. To collect such data one has to compare the new technologies to existing ones.

The test subjects in studies such as this are usually students that are trained for a short time and then conduct the experimental tasks for a couple of hours. However, the results say much more about the performance of experts in real situation if these same experts are part of the study. The biggest problem with this approach is the cost of using trained experts. It is hard to motivate companies to spend a lot of time/effort/money to let their employees take part in a study where they a small toy system and are not available for other tasks.

Most case studies are also done with artificial systems or systems of a restricted size which limits the generalizability of the conclusions that are drawn from them. The problem of using real systems is that it can interrupt the workflow of a project if they are conducted during the development process.

4.1 Study Goals

In this case study we compared MBT to manual testing which is still commonly used since it has advantages (e.g. low initial cost, high flexibility) that make it appealing.

In order to evaluate the costs and benefits of MBT in an industrial setting, we designed a case study with the following goals. We wanted to study how effective (number and type of issues found) and how efficient (effort spent) manual testing and MBT with FSMs are.

Detected issues were categorized in order to determine which approach tends to do better per category. The effort was also categorized to analyze where each approach spends its effort. The two approaches were applied to two versions of the system under test. The first version was an initial release of the system that had been rolled out to the customers for evaluation. The second version was a redesigned version of the original system where the graphical user interface (GUI) of the system had been changed and several issues had been fixed. By applying the two testing approaches to two versions we could also analyze how the two approaches handle regression testing. In addition, we collected lessons learned and possible points for improvements of the approach and of future studies.

4.2 Study Design

This section describes the design decisions of our study. We decided to use people who are already experts on a certain technique and let them perform a task they are comfortable with
instead of using students who only have a couple of hours to learn a formerly unknown technique. Naturally, very few such studies exist since it is difficult to motivate companies to spend a lot of time/effort/money to let their employees take part in study where they all test the system in parallel. For this reason the number of test subjects in this study is also limited.

Another problem with these types of studies is that they can interrupt the workflow of a project if they are conducted during the development process. In order to minimize the invasiveness of this study, it was conducted in parallel to development process at GNSI. This means that GNSI performed their regular development process and tested the system. At certain intervals the system was set aside for the two test subjects, who were not part of the regular development project at GNSI, to perform their testing and collect the data.

The choice of testing a real system under development gives more insights into real problems that one encounters when a system needs to be tested. One example we encountered was related to the technical limitations of the test execution software that we had to overcome in the MBT approach. However, there are downsides to using real systems that are in production. It is difficult to isolate them and have full control since the testers had to share it with other testers and customers doing training sessions. In the case of our study we encountered the issue that data entered by the MBT tester was not removed properly and was accidentally used by the manual tester. Also real life events can interrupt long time studies running over several weeks. For example, we had to coordinate our testing so that we did not interfere with the training sessions that were conducted with future users of the system.

The study was carried out by two management teams and two testing teams. The GNSI management team coordinated all efforts on the GNSI side, appointed the manual testing team, provided all artifacts (requirements, URL to test system etc.) and selected the system and versions to be tested. The Fraunhofer management team coordinated all efforts on the Fraunhofer side and appointed the testing team. The GNSI testing team consisted of one manual tester from GNSI that used the regular testing practices from GNSI. The Fraunhofer testing team consisted of one model based tester together with a student intern from Fraunhofer that used the FSM based MBT approach described earlier. The testing teams had no direct contact with each other during the testing process. The teams had access to identical documentation of the SUT and could also contact the development team with questions about the system and its requirements, which they did on a couple of occasions to clarify how the system was supposed to work. Each team member logged the effort spent for later analysis. The effort was recorded in two different sets of categories, one for each approach, since the sub tasks of the approaches were not identical.

Each team was instructed to apply their approach on the two versions of the system and report all issues that they detected. No time limit was given; instead each team used their own discretion to determine when to stop each testing task while recording the time they spent. The reported issues were later collected, analyzed and categorized.

Part of the testing was observed by a Fraunhofer researcher who also interviewed the two subjects on their background and their qualifications. The observation was conducted in order to get a deeper understanding of the two different testing processes and where time was spent. Based on this data we collected lessons learned about the testing processes and identified improvement points in the workflows.

The system to be tested and the different versions were selected entirely by the GNSI study management group. Thus the two testers did not have any influence on what version of the system to test next would be. The versions were tested as they were developed by the development team so to mimic a regular software development project. However, the system had to be delivered according to a contract and a late delivery could not be tolerated by the company. Since the testing study required that the two testing teams were relatively synchronized in their testing, it was decided that this entire testing study would be conducted in parallel to the regular testing effort of the development and testing teams.

The characteristics of the system were similar to the one from the pilot study and similar to other systems in production at GNSI, so we decided that it would be a good fit for this comparison.

4.3 The System Under Test

The SUT is a web based data collection and review system for papers about food borne illnesses. The first version of the system that was tested is an extension of a larger application based on Oracle’s Portal framework. The second version was a standalone version of the extension that was not based on the Portal framework.

The system has two different kinds of users. Users, who can only enter data into the system and review/edit their own entries and Reviewers, who review and approve/reject the entries of the Users.

Users can enter data on a web form and choose pdf attachments. All entries of the user can be seen on an overview page from which the user can go to a review/edit screen of a specific entry or delete it. Reviewers can see the entries of all the users in a review specific overview page and also open them in the review/edit screen. They see a slightly modified version of the screen with fields for commenting and buttons to approve or reject an entry. In addition to the user specific overview page there is also an overview page that shows all the approved data if it fits into a specific category.

4.4 Artifacts available to the testers

The following artifacts were made available to the testers:

- Two use cases describing the data entry and data editing functionality of the system. The goal was to test these two use cases and parts of the systems that they affected. E.g. one use case mentions that the data will be placed on a certain part of the page to be accessible for the user then we would test that it will show up there.

- A list of requirements of the system. Only some of them are relevant to the use cases so each tester had to decide which ones were relevant.

- Access to the developers to clear up any questions about the requirements or the system.

- The system itself. Each tester had access to the GUI of the system and was given different user accounts for the testing. The testing was done as black box testing and the testers had no access to the source code.

5. The two testing processes

5.1 Manual Testing

The manual testing process consists of several possible steps. Not all of them are necessary; some can be left out or be adapted. In
the first step, the manual tester tries to understand the system under test. For this, the tester analyzes the documentation of the system (e.g. requirements, use cases), does exploration of the system and/or discusses the system with its developers. This step is necessary to understand how the system is supposed to work and to identify which parts of the system need to be tested.

In the next step, the tester creates test cases for the SUT that are based on the information from the system understanding step. The test cases where created in an excel sheet that contained the steps that the tester should execute and the expected results of the execution. For example navigating to a specific page and inspecting the elements of that page.

The test cases in this case are a set of manual instructions on how to test the system. The tester uses them as a guide on how to test the system. During this manual execution the tester observes the system behavior, analyzes and reports issues that are detected. During the execution of the test cases the tester also looks at other aspects of the systems and tries out other instructions besides the one in the test cases. This is usually based on the intuition of the tester and not all of these tests are recorded for later reuse.

5.2 Model Based Testing
The MBT approach that was used in this study uses FSM models to represent the expected behavior of the SUT. The approach is similar to the one described in [4]. Figure 2 shows a simplified example model to enter and remove user. The models consist of transitions, which represent requests (or stimuli) to the SUT, and states containing information about the conceptual state that the SUT is expected to be in after the requests have been executed. In the example FSM the transition add_entry relates to an action that enters data into the system and the state Entry_added indicates that the system is expected to contain a data entry now.

This model is the basis for generating test cases. A test generation tool uses it as an input and traverses the model by following the transitions in the model until a stopping criterion is reached (e.g. reaching the state labeled Exit). The output of this step is a suite of abstract test cases in that each test case is a path from the start state to the exit state of the model, together with the expected behavior of the SUT for each action. By mapping the states and transition to a test execution framework these abstract descriptions can be translated into concrete automatically executable test cases that can test the system without assistance of the user. This is done by mapping the abstract actions and reactions to concrete implementations in a programming or scripting language. An example how this looks conceptually can be seen in where the transitions add_entry and remove_entry and the states Entry_Added and Entry_Removed are mapped to Java code that will interact with the system under test.

<table>
<thead>
<tr>
<th>Transition/State name</th>
<th>Associated Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_entry</td>
<td>testingFramework.goToDataEntryPage(); testingFramework.enterData();</td>
</tr>
</tbody>
</table>

Figure 2: The Model Based Testing Workflow
Instead of writing test cases directly like the manual tester, the tester models the functionality of the system and the expected behavior using a finite state machine model. The tester has to choose the granularity at which he will model the system. E.g. the transition add_entry could be split up into a several transitions. Choosing the right granularity is a very important aspect and requires experience from the tester. Choosing a too fine-grained model causes more work in test infrastructure creation and the model has to be updated more often if the SUT changes. Choosing a granularity that is too abstract and the model is unable to detect issues in behaviors that are lost in the abstraction. The model encodes the expected behavior that the tester derived in the previous step. In a FSM the transitions are functionalities of the system under test (e.g. add a data entry into the system) and the states are the corresponding expected behaviors. Error! Reference source not found. shows an example of how a FSM model looks like. The transition with the label add_entry represents the action of entering data into the system under test. The corresponding expected behavior is encoded in the target states of the transition. The label of the state in our example is called Entry_Added which is the expected behavior after we entered data into the system.

Figure 1: Example of a FSM based testing model
A test case in its simplest form is a traversal through the model from the start state to the point where the stopping criteria is met. The model based testing tool used in this study was Jumbl[10]. The stopping criterion for Jumbl is the state labeled Exit. Jumbl will either randomly traverse the model until the exit state is reached or the user can instruct Jumbl to produce as many test cases from start to exit as are needed to cover all states and transitions of the system. The rest of the section will describe a more detailed, step by step walkthrough through the MBT process that was used in this study.

The first step is similar to the manual testing approach. The tester tries to understand the system by using the given documentation (e.g. requirements, use cases), by using the system and by talking to the developers of the system.
The test generation and test execution process is automated and requires nearly no human effort (besides the setting of some parameters). The result of the test execution is a list of executed test cases together with information whether or not and where the test cases failed. This result list is analyzed manually by a test engineer to determine if the results are correct or not. This part needs the tester’s interaction by judging whether a test case that did not pass constitutes a bug in the SUT and what actually caused it. This can involve replaying a test case or manually reproducing parts of the test case to analyze what caused the bug. Furthermore, it is important to note that there can be false negatives due to bugs in the test execution framework.

This process is iterative allowing one to start with a small portion of the system and later add more functionality either by refining the existing parts of the models or by extending the models to cover more features.

For testing the application in this paper the test execution framework was based on Selenium\(^5\) in conjunction with JUnit\(^6\). Selenium can be used to automate the testing of web systems. Selenium’s Java API was used to interact with the SUT and JUnit to execute the generated tests.

6. Results and Observations

6.1 Issue Analysis

6.1.1 Issue Results

Each issue that was reported was classified according to the following issue classification scheme. The scheme was defined bottom-up, i.e. it was based on the issues that were found.

- **Business Logic Issues**
  - These entail problems in the workflow of the system. For example the user enters data and then wants to press the save button but the button is not responding.
  - E.g. Functional Issues

- **Field Validation**
  - Field validation issues are issues that are caused by missing or incorrect field validators. For example entering a certain number of characters into a field can cause the system to crash.
  - E.g. Field Length violations are not handled correctly

- **Naming Discrepancies**
  - These are deviations between the naming of labels in the documentation and in the implementation of the system.
  - E.g. “Lab or Organization” instead of “Organization”

- **Field Discrepancies**
  - These are deviations between the fields that are described in the documentation and the fields in the actual system.
  - E.g. Extra/Missing Fields

- **Usability Issues**
  - These are general usability issues that are not directly interfering with the functionality. For example a back button on a multi-page form that is not working. Or layout breaks caused by certain input data.
  - E.g. Broken Layout

  - E.g Buttons to abort a task missing the user could only move forward

For testing the application in this paper the test execution framework was based on Selenium\(^5\) in conjunction with JUnit\(^6\). Selenium can be used to automate the testing of web systems. Selenium’s Java API was used to interact with the SUT and JUnit to execute the generated tests.

Table 2: Issues in the original version of the system

<table>
<thead>
<tr>
<th>Category</th>
<th>Overall</th>
<th>MBT</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Logic</td>
<td>22</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>Field Validation</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Naming Discrepancies</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Field Discrepancies</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Usability</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>42</td>
<td>26</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 3: Number of distinct issues in the original version of the system

<table>
<thead>
<tr>
<th>Category</th>
<th>Overall</th>
<th>MBT</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Logic</td>
<td>22</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>Field Validation</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Naming Discrepancies</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Field Discrepancies</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Usability</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

\(^5\) http://docs.seleniumhq.org/
\(^6\) http://junit.org/

---

Table 1: Example of a mapping between states/transitions and the corresponding code that interacts with the SUT

<table>
<thead>
<tr>
<th>Testing Step</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove_entry</td>
<td>testingFramework.goToDataOverviewPage(); testingFramework.deleteData();</td>
</tr>
<tr>
<td>Entry_Added</td>
<td>assertTrue(testingFramework.isEntryAdded());</td>
</tr>
<tr>
<td>Entry_Removed</td>
<td>assertTrue(testingFramework.isEntryRemoved());</td>
</tr>
</tbody>
</table>
Table 4: Issues in the new version of the system

<table>
<thead>
<tr>
<th></th>
<th>Only MBT</th>
<th>Both</th>
<th>Only Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Logic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Validation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naming Discrepancies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Discrepancies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>36</td>
<td>28</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 5: Number of distinct issues in the new version of the system

In the new version of the system, the overall number of reported issues amounted to 36 of which MBT reported 28 and Manual reported 17, see Table 4. MBT still reported more issues related to Business Logic and Field Validation but for Usability issues, the two approaches reported almost the same number of issues (6 vs. 5) with a great overlap.

It is important to note that the issues related to Naming and Field Discrepancies that were reported for the original version were still in the new version, but were not reported by the manual tester this time.

The issues that were reported for the new version are still very distinctly distributed among the two approaches. 19 issues were reported of the 36 issues were reported by the MBT approach and 8 by the manual. The 9 remaining issues were reported by both approaches. See Table 5.

Table 6: Number of total issues in the two versions

<table>
<thead>
<tr>
<th></th>
<th>Only MBT</th>
<th>Both</th>
<th>Only Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Logic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Validation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naming Discrepancies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field Discrepancies</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>52</td>
<td>36</td>
<td>29</td>
</tr>
</tbody>
</table>

Table 7: Number of total distinct issues in the two versions

Since some of the issues were fixed from one version to the next and some new issues were reported, we summarized all reported issues (see Table 6 and Table 7). It shows the total issues found in the two versions this includes all issues found in the first version plus all newly reported issues in the second version of the system under test. MBT found more in the Business Logic and Field Validation category, whereas the manual tester reported more issues in the Naming and Field Discrepancy classes. It can also be seen in Table 6 that the total number of issues reported by both approaches is very similar whereas Table 7 shows that the issues that the two approaches reported are very different.

### 6.1.2 Issue Discussion

One aspect that is immediately visible from the issues that were collected is that the two approaches, even though they found a comparable number of issues, only a few of them were found by both approaches. The mostly disjoint issues indicate that the testers focused their efforts on different aspects of the system and had different test objectives. The manual tester for example looked for discrepancies in the field names (there was a specific manual test case for it that had all the expected field names listed). The MBT tester focused mostly on functional issues. Adding a check for field names could be added in the model but it does not fit well into the FSM approach that models the expected usage and behavior of the SUT. Creating a simple Selenium script to check this would have been the best way to deal with this issue from a test automation point.

Also since the models for MBT were focused on functionality issues (Business Logic, Field Validation) it was expected that MBT would perform well in these categories. The same is true for field validation when we develop models we make sure that all corner cases for inputs are considered (e.g. long inputs, inputs with special characters, no inputs). Automated testing is very well suited for this because a large number of test cases are needed to cover all necessary scenarios. A human would take much more time. Also a human might make mistakes in entering the test data therefore invalidating the test case by accident.

### 6.2 Effort Analysis

#### 6.2.1 Effort Results

For the effort analysis each study participant reported their time as part of pre-defined tasks.

- System Understanding/Exploration
  - Analysis of the documentation consisting out of use cases and a list of requirement. Inquiries to the developers, exploration of the system under test. Any issues discovered in this phase were reported. This task was carried out by both testing groups.

- Modeling
  - Creation of the test models based on information gained in the requirements elicitation. This task is only necessary for MBT.

- Implementing Test Infrastructure
  - Automatically generated test cases need a test infrastructure to make them executable on the system under test. This task is only necessary for MBT.

- Test Case Development
  - Writing test cases based on the requirement elicitation information. This task is only necessary in manual testing.

- Test Execution and Issue Analysis
  - This is the manual task for testing. It entails the execution as well as the analysis of the result of the test case since the two tasks are entwined in manual testing and cannot be reported separately. The manual tester analyses and reports possible issues while executing the test cases.

- Issue Analysis
  - Unlike manual testing the test execution is done automatically and the tester just analyses the results of the execution in form of a test execution report that
shows which test cases failed or passed and uses this as a starting point for investigations into possible issues.

Since the workflows of the two approaches are different the categories are also different and a direct comparison category by category is not possible. We therefore grouped the above listed tasks into three high level tasks. These high level tasks are Initial Effort, which includes everything up to the point where the actual testing can start, and the two testing phases for the two different versions of the systems that were tested.

<table>
<thead>
<tr>
<th>Task</th>
<th>MBT</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Understanding/Exploration</td>
<td>16.5</td>
<td>16</td>
</tr>
<tr>
<td>Modeling</td>
<td>24</td>
<td>N/A</td>
</tr>
<tr>
<td>Implementing Test Infrastructure</td>
<td>87</td>
<td>N/A</td>
</tr>
<tr>
<td>Test Case Development</td>
<td>N/A</td>
<td>16</td>
</tr>
<tr>
<td>Total</td>
<td>127.5</td>
<td>32</td>
</tr>
</tbody>
</table>

**Table 8: Initial effort in person/hours sorted by category**

Table 8 shows the initial effort that the two teams spent to get ready for the testing phases. Both teams spent around 16 hours on requirement elicitation, which includes analysis of the requirements, questions to the development team and system exploration. Modeling is only done by the MBT team which spent 24 hours to create the testing models of the SUT. Implementing the test infrastructure is also a task that is not carried out by the manual tester. It took the MBT team 87 hours to implement the necessary functionality to execute the tests automatically. Test Case Development is only carried out by the manual tester since the test cases for MBT are derived automatically from the testing models. Creating the manual test cases took 16 hours. The total initial effort for MBT was therefore 127.5 hours and for the manual tester it was about one fourth of the time with 32 hours.

<table>
<thead>
<tr>
<th>Task</th>
<th>MBT</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Execution and Issue Analysis</td>
<td>N/A</td>
<td>26</td>
</tr>
<tr>
<td>Issue Analysis</td>
<td>4</td>
<td>N/A</td>
</tr>
<tr>
<td>Total</td>
<td>4</td>
<td>26</td>
</tr>
</tbody>
</table>

**Table 9: Effort for testing the first version of the system in person/hours**

The two teams developed the tests for the initial version of the system so there was no effort necessary to adapt to it. Testing the initial version took the MBT tester a total of 4 hours and the manual tester 26 hours. The tasks were not identical as can be seen in Table 9. The manual testing task consists of test execution and issue analysis. Since the two are highly entwined it is impossible to log how much time was spent for each part. The MBT tester just had to do the issue analysis since the actual test execution was done by the automated test execution framework which just takes computing time.

<table>
<thead>
<tr>
<th>Task</th>
<th>MBT</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapting test cases for new version</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Test Execution and Issue Analysis</td>
<td>N/A</td>
<td>7</td>
</tr>
<tr>
<td>Issue Analysis</td>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

**Table 10: Effort for testing the second version of the system in person/hours**

For the second version the manual tester and the MBT tester applied the same test cases that were developed initially on a modified version of the system. In this new version, the GUI of the web system was changed drastically. Adapting the test execution framework to these changes took 6 hours. The manual test cases did not have to be adapted since they are independent of any test execution framework. The test execution and issue analysis took the manual tester 7 hours. The issue analysis with automated test execution took the MBT tester 2 hours. The total effort for testing this version was 8 hours for MBT and 7 hours for manual. The overall effort of all tasks was 139.5 hours for MBT and 65 hours for manual testing as can be seen in Table 11: Overall effort.

**Table 11: Overall effort**

For the second version the manual tester and the MBT tester applied the same test cases that were developed initially on a modified version of the system. In this new version, the GUI of the web system was changed drastically. Adapting the test execution framework to these changes took 6 hours. The manual test cases did not have to be adapted since they are independent of any test execution framework. The test execution and issue analysis took the manual tester 7 hours. The issue analysis with automated test execution took the MBT tester 2 hours. The total effort for testing this version was 8 hours for MBT and 7 hours for manual. The overall effort of all tasks was 139.5 hours for MBT and 65 hours for manual testing as can be seen in Table 11: Overall effort.

**6.2.2 Effort Discussion**

Both testers took about the same amount of time to analyze the requirements. This is not unexpected since the tasks in this phase were the same for both of them. Most of the effort for the MBT approach was spent on implementing the testing infrastructure. About half of that time was spent because of issues that the MBT team had to solve with the test execution framework. These included learning additional technologies which are one-time efforts and could be reused in similar systems with a fraction of the effort. In another task the automated test cases had navigate to a page and inspect a complicated table structure on the webpage in order to judge its correctness. The program used selenium and java to extract the structure from the webpage and put it into a Java class that could be queried in order to judge the correctness of certain data entries. As with the additional technologies these tools can be reused in other testing tasks. The analysis of this structure yielded 5 errors in the Business Logic of the SUT. Aside from these issues the implementation of the test execution framework was relatively simple, but due to the large number of elements on the web page it still took around 40 hours to implement and debug.

The changes in the system between the two tested versions did not affect the functionality of the system but the user interface. The manual tester did not have to change their test cases to adapt them to the new version since they were on a higher level of abstraction than the changes in the system. For example the manual test case just mentions going to a certain page and does not contain detailed instructions about which buttons to click to get there. The same was true for the models that were developed for MBT they were abstract enough and did not have to be changed.

However, test execution frameworks for web testing can be brittle. They rely on identifiers to select elements on the web page. Most of these identifiers changed in the updated version and had to be replaced as well as most of the navigation through the
system. The impact of this can be seen in Table 10 which shows the effort data of adapting the testing framework so it could test the new version of the web page. This shows that humans can adopt much better to changes on GUI’s than automated frameworks and they are not easily thrown off by changes.

The data shows that the initial investment for manual testing was about ¼ of the MBT effort. This is a much higher number than in the effort model presented in [4]. In their model the initial investment for MBT was lower than for manual testing. However it is hard to compare those numbers directly since the characteristics of the SUT in their model are very vague. We just know that it they simulate a small application with roughly 15k lines of code. It is unclear if it is supposed to be a GUI based system or a system based on a command line interface. But these characteristics can have a high impact on the necessary effort. From our experience with other systems we know that implementing a testing framework for a GUI based systems often requires more effort than for an API based system.

Our interviews and observations also indicate that there are overlaps in the tasks that the two teams had to do. Mainly in the system exploration and requirements analysis. Furthermore the data shows that both testers spent roughly the same amount of time here.

In addition to just presenting the raw issue data we also put them into relation to each other. We wanted to see how many hours were spent on average to find an issue. Table 12 shows the effort in hours per issues for MBT and for the manual testing.

<table>
<thead>
<tr>
<th></th>
<th>Effort</th>
<th>Issues</th>
<th>Effort/Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>65</td>
<td>29</td>
<td>2.24</td>
</tr>
<tr>
<td>MBT</td>
<td>139.5</td>
<td>36</td>
<td>3.86</td>
</tr>
<tr>
<td>MBT*</td>
<td>139.5 (-40)</td>
<td>36</td>
<td>2.76</td>
</tr>
</tbody>
</table>

Table 12: Effort per issue overview

On average the tester using MBT spent 3.86 hours per reported issue whereas the manual tester reporter an issue every 2.24 hours. We also added a third comparison point named MBT* that excludes the one-time cost to develop the solutions in the testing framework discussed earlier in this section. This comparison shows a slight lead for the manual testing task. However, this number is expected to shift in favor of the MBT process if more iterations of the testing process are needed.

7. Lessons Learned and Discussion

Applying new technologies always poses challenges and there is often room for improvements. One aspect that was interesting was where these challenges are in applying MBT on a real system. Another point we wanted to study was the existing manual testing process. Are there ways to improve and enhance it? Besides these two points we also learned several lessons on how to conduct testing studies in an industrial setting that we would like to share as well.

We compared MBT with manual testing and therefore we tried to test the system using only MBT. Furthermore we created only one large model for the system. The whole model, as well as slices of it was used to generate tests. In retrospect this was not the optimal way to test the system. Instead we should have divided the system into complex and non-complex parts and created smaller specialized models for the parts with a higher complexity and used simple automation scripts for the parts without high complexity.

An example of this was the data entry page of the system. It has many simple text fields that were grouped together in the model by topics (e.g. address) and for each topic area the model had transitions for the different boundary conditions (e.g. field length, special characters). This made the model and the resulting test cases very large. This level of fine grained testing could have been more effectively handled via an automated test scripts that would have the entire range of boundary input values and would just test this aspect of the page, thus making the model and the resulting tests less complex. The rest of the model should have focused on the more complex aspects of the system to play out the strengths of MBT more. For example the data entry page had certain complex aspects to it besides the text fields. The user can choose values from two drop down lists and add them to a list that can hold a certain number of entries. If the maximum number is reached the system will show a popup message to inform the user that no more entries are allowed. The user can also remove entries after they were added. This behavior was captured in the model but it was not sliced properly and therefore not covered well during the test generation. A smaller model with a focus on these complex aspects would have been easier to slice and would have probably caught an issue that MBT missed. Another aspect that made the testing harder was the fact that our tool chain had no built in support for model slicing and only two model traversal strategies. We worked around the model slicing by creating a large model that had all the functionality and then created copies of this model were certain parts were omitted. This worked well enough but it added a maintenance problem; if the original model had to be changed the copies had to be updated as well. Ideally we would want more control in the test generation so that the test generation strategy defines the slice. For example we would want to specify a list of transitions that each test case has to take and then start a random traversal from this point. Another feature for slicing would be to block certain transitions so the test generation does not take certain paths. However, the tool chain did not support such features at the time.

The length of the test cases became an issue in the analysis of the test results. The problem of a failing test case is to figure out which parts of the test case were responsible for its failure. The problem gets even harder because the test execution is fully automated the tester is not aware of what actually happened in the programs he just sees the results of the test execution. The manual tester has a big advantage there. The manual tester sees all the responses of the system to his inputs and it is therefore much easier for him to identify the cause of the issue. The automated tester might have to retrace some of the steps in the test case before being able to identify the cause. Both the model specialization and the advanced model slicing would have resulted in more but much shorter test cases and helped make the test analysis easier.

One of the points that the observer mentioned was the disconnectedness between the test cases and the model. In the analysis of the test case results the tester did not use the model anymore. Asked about that fact, the tester replied that it was hard to go back and forth between the model and the test case, because there was no direct link in the test case that pinpointed the location in the Model. This fact can be overcome by adding information about the state and transitions into the generated tests. For MBT techniques that are based on extended FSM’s it is also important to add information about the state variables and their values into the test case.

Using Selenium to drive the test execution caused JUnit to report some false negatives. In some cases it did not recognize objects
(e.g. a button) properly and caused the test to fail. If the test case was run again it would recognize it. The button in question however was always available and should have been detected. This causes additional work for the tester since he needs to analyze if it is an issue in the test execution framework or in the SUT.

Based on the observations in this study we evaluated other MBT tools and chose GraphWalker as a successor to the previously used tool called JUMBL. GraphWalker gives the tester more options in the test generation process and adds additional features like extended state machines that can make models more efficient. Also the strategy for model creation changed slightly, in the revised process the tester comes up with the smallest possible models for different parts of the system. The initial set of test cases are therefore very specific to a limited part of the system, this makes the analysis of the test cases and the identification of issues easier. From this initial set of models we refine the model of the system by adding additional behaviors or by combining models.

Another finding of the study was that some features of the systems would have been easier to model if the MBT tool would have supported extended finite state machines (EFSM). An EFSM is an extended version of a FSM that adds variables and guards. The variables can be used as counters for example and the guards can be used to open or close transitions if the variables have a specific value. The data entry form had a list where the user can add entries until a certain limit is reached. This kind of behavior can be modeled better in EFSMs than in FSMs.

One of the biggest obstacles in the adoption of MBT is the necessity for training of the testers. MBT requires a different mindset than traditional testing. They need to learn how to abstract the system and design the models. Furthermore it requires programming knowledge to implement the test execution framework for a model, which are harder to develop than regular test automation scripts. Testers often have good domain knowledge and knowledge about test design but not about modeling for test generation.

The issue and effort data we collected does not give a clear answer to the question which technique is better. It all depends on the system under test as well as on testing goals. Our experience also shows that there is not necessarily a clear line between MBT, automated testing and manual testing. For example even a tester that applies MBT does some manual exploration of the system where he analyzes the system and decides which aspects of the system are captured in the model. Issues found during this stage are of course also reported. Some parts of the system can also be hard to analyze automatically. Examples of this are page layout issues on web pages which are very hard to verify automatically but easy to identify by a human tester.

Also existing manual tests can be a great asset for the model based tester and can be incorporated into the model. Based on a few test cases it is not hard to create an initial model and quickly generate tests that are valid permutations of these test cases.

Manual testing can be a repetitive and can be a very stressful task over long periods. For GUI- and Web-based systems capture replay tools can help make the tedious parts of data entry easier and the whole process faster. Also capture replay tools can be used by testers that do not have a programming background.

The study compared the issues that were found and the effort that was necessary to find them. One more interesting measurement would be how much of the system each tester covered. This would give us a measure of confidence for each approach. Taking coverage measurements was not feasible in the test system since we had to share it with other users.

8. Threats to validity

The discussion about threats to validity is based on the model of Wohlin et al [11]. They define four classes of threats to validity, namely threats to internal, external, construction, and conclusion validity.

8.1 Threats to internal validity

Threats to internal validity are caused by factors that were not considered but might have influenced the results of the case study. Having no complete oversight about the test subjects during the study poses a risk to the internal validity. The study participants and employees were instructed not to give out information about the testing process of the two groups and the two groups themselves did not communicate during their testing sessions. However, one of the participants might overhear someone else talking about it in the office. It is of course not possible to fully control an aspect like this if the testers are not monitored at all times. Nevertheless, based on the differences of the detected issues we saw from both testers we do not believe there was any interchange between the groups during the testing session.

One part of this case study was the evaluation of the results of the two testing teams to do a comparison. Ideally this evaluation should have been carried out by a party that was not involved in the testing process, but the analysis needed a high level of system understanding and the third party person would have had to analyze the system as well to understand the presented results. Due to constraints in the budget we decided that the MBT tester should start with the comparison after the first iteration of the testing was finished. At this point the modeling process and the test generation process was already finished. So the test cases in the MBT part were fixed and the information from the manual testing process could not have influenced the outcome of the second session anymore.

In one case the MBT tester did not clean out his data between the testing sessions properly. The data that was entered was not well formed and the system should not have accepted it as a valid input. Any attempts to further work with this data entry caused system crashes. The manual tester used this data entry in the testing process and reported the crashes that working with it caused as issues. Because they were just symptoms of the invalid data they were not counted in the issue comparison.

8.2 Threats to external validity

Threats to external validity cover aspects that might influence the generalizability of the case study. For example, could another person repeat the case study and come to the same conclusions, or could you repeat the case study with a system in a different domain and come to the same conclusions?

The candidates for the case study can pose a risk to the external validity. Using students for example can bias the results since they might not be as effective at applying the approach as trained persons. We therefore chose participants that were already trained in using their approach and had experience applying it to other software systems.

The system we tested was web-based so we believe the conclusions and lessons learned in this case study are valid for GUI based systems in general, as long as there exists automation technology for it. Some of the conclusions might also be valid for
other kinds of systems (E.g. API based) but further study is needed on this topic.

8.3 Threats to construction validity
Threats to construction validity assess if the correct measurements were used in the case study. In this study we used the effort in hours and the number of issues as basic measurements. These are basic measures that are not derived and therefore a good indicator for the efficiency and effectiveness of the two approaches.

Since it was infeasible to have someone look over the shoulders of the test subjects for the duration of the study, they had to report their hours themselves. We believe this is a general trade-off in most real life studies that cannot be avoided. But in order to put some checks in place one of our scientists observed the testers for a limited time to get an understanding for their process and to be able to judge if the numbers they reported are feasible.

8.4 Threats to conclusion validity
Threats to conclusion validity cover issues that affect the ability to draw the right conclusions from a case study. In this case study we used real employees instead of students which is very costly and can interrupt the regular work schedule of a company. We therefore had to limit the number of participants to only one tester for each group and to two test iterations. But as shown in this paper there are valuable lessons that one can learn, even with a few test subjects, that one might not be able to learn in a more controlled student-based study. Real employees also know the process better than students and do not need training. They are already experts in their field which will most likely yield different insights than those you can get from beginners. Observing testers under real life scenarios is very valuable. An outside observer who is not an expert, but has knowledge about, both approaches is not biased and can identify potential issues that the testers themselves do not see. However, the observations can influence their behavior and therefore the outcome of the study. Therefore, we only observed the study candidates for a limited amount of time.

The study was done with only one system, which restricts the generalization of the study. Nevertheless it can be compared to previous studies and be a valuable data point for comparison with future studies.

9. Conclusion and future works
The original question was: Does Model Based Testing beat Manual Testing? Our answer is: Yes and No. It depends on many different circumstances and factors. MBT is more thorough than manual testing and can find more issues. However, it is not equally well suited for all testing tasks.

MBT is good at finding functional issues and making sure that corner cases of data inputs are considered but it is not well suited for the verification of the layout of a graphical user interface for example. Here it is often still easier and cheaper to have the human tester verify these aspects of the system. Furthermore, the MBT development process still has a strong manual component when it comes to the analysis and exploration of the system. So the manual testing process cannot be replaced completely.

Additionally, the process of designing and implementing the test infrastructure for MBT requires programming skills whereas the manual tester of a web based system does not need any software development skills.

Based on the experiences of this case study we identified lessons learned that can improve both the MBT and manual process. We believe that a hybrid approach that takes the strengths of both approaches and combines them would be the most effective. We also started implementing and evaluating the improvement ideas for the MBT process and already built some of them into our process.

Based on this work we are planning to do a follow up study with more participants where we want to validate the results of this study as well as evaluate some of the improvements we suggested for MBT. In addition to that the study will also try to evaluate the cost of teaching MBT to the participants.

10. REFERENCES


