Efficient transformation from general flow into a specific test case in an automated testing environment

Andreas Oskarsson
This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author: Andreas Oskarsson
Address: Furutåvägen 19A, 352 54 Växjö, Sweden
E-mail: andreas-oskarsson@home.se

External advisors:
Thomas Pihl & Olof Carlsson
WM-Data Consulting
Address: Smedjegatan 12, Box 159, 351 04 Växjö, Sweden
Phone: +46 (0)470 – 77 16 00

University advisor:
Olle Lindeberg
Department of Software Engineering and Computer Science

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/ipd
Phone : +46 457 38 50 00
Fax : +46 457 271 25
ABSTRACT

SIMON is an automated testing application developed by WM-Data Consulting in Växjö, Sweden. Previously the test cases, called BIFs, run by SIMON to test the applications under test has been written manually in a very time consuming manner offering no protection against errors in the structure or misspellings. This thesis investigates a replacement to the manual method when creating the BIFs; my own developed application called the BIF-Editor. The usage of the BIF-Editor guaranteed correct syntax and structure and made the creation of the BIFs faster, but did it increase the quality of the BIFs?

So to evaluate the BIF-Editor, the quality regarding path coverage of BIFs manually created was compared with BIFs created during the same elapsed time using the BIF-Editor. This evaluation showed that the usage of the BIF-Editor increased the quality of the BIFs by making the creation safer, but primarily faster which enabled the user to produce more BIFs than previously possible resulting in a raised path cover.

Keywords: Automated testing, path coverage, software testing, action research
ACKNOWLEDGEMENT

I would like to thank the group at WM-Data Consulting in Växjö, with Thomas Pihl in charge, for letting me do my thesis there and especially my supervisor at WM-Data, Olof Carlsson for all the help and guidance with the creation of the BIF-Editor.

I would also like to thank my supervisor Olle Lindeberg, Department of Software Engineering and Computer Science, at the Blekinge Institute of Technology for the feedback and help along the way.

However, most of all I would like to thank myself for doing all the work…
CONTENTS

ABSTRACT ..................................................................................................................................................... I

ACKNOWLEDGEMENT ............................................................................................................................. II

CONTENTS ........................................................................................................................................................ III

1 INTRODUCTION ..................................................................................................................................... 1

1.1 BACKGROUND .................................................................................................................................... 1
1.2 OBJECTIVES ......................................................................................................................................... 2
1.3 HYPOTHESIS ....................................................................................................................................... 2
1.3.1 PATH COVER ................................................................................................................................ 3
1.4 DELIMITATIONS .................................................................................................................................. 3

2 SOFTWARE TESTING ............................................................................................................................ 4

2.1 HISTORY OF SOFTWARE TESTING ............................................................................................. 4
2.1.1 PURPOSE AND QUALITY OF TESTING ....................................................................................... 5
2.2 AUTOMATED TESTING ..................................................................................................................... 6
2.2.1 TESTING INTERFACES .............................................................................................................. 8
2.3 GUI TESTING ..................................................................................................................................... 10
2.3.1 RECORD/PLAYBACK TEST SYSTEM ....................................................................................... 10
2.4 SIMON .................................................................................................................................................. 11
2.4.1 BUSINESS INFORMATION LOGIC ............................................................................................ 11
2.4.2 BUSINESS INFORMATION FILE ............................................................................................... 12
2.4.3 BIF-EDITOR .................................................................................................................................. 14

3 EXTENSIBLE MARKUP LANGUAGE ................................................................................................. 17

3.1 MARKUP LANGUAGES ..................................................................................................................... 17
3.2 HTML .................................................................................................................................................. 17
3.3 XML .................................................................................................................................................... 18

4 THE CREATION OF THE BIF-EDITOR ............................................................................................... 19

4.1 ANALYZE, DESIGN AND IMPLEMENTATION ............................................................................... 19

5 EVALUATING THE BIF-EDITOR .......................................................................................................... 20

5.1 ACTION RESEARCH ......................................................................................................................... 20
1 INTRODUCTION

This report is my Master Thesis in Computer Science conducted for the Department of Software Engineering and Computer Science at Blekinge Institute of Technology (BIT) in Ronneby, Sweden, during the spring of 2003. My name is Andreas Oskarsson and I was enrolled in the Information system program of 1999.

This thesis target readers with interest in the development of SIMON, readers with a more general knowledge within the domain of automated testing or readers with interest of system development in general within the domain of Computer Science.

The comprehensive reason for this thesis was my desire to perform a “real” task for a contact within the industry, instead of performing another “fictional” one like I did when conducting the Bachelor Thesis. My belief was that a practical research assignment in the industry about a real task offered me as a student more knowledge, but also experience, than any of my own pretended inventions would.

Therefore I contacted Thomas Pihl at WM-Data Consulting in Växjö, Sweden, and after an interview we agreed to cooperation about a thesis regarding their automated testing system SIMON.

1.1 Background

SIMON is an XML-based automated testing application developed by WM-Data Consulting in Växjö, Sweden. It is about two years old and has so far been used to test various applications developed by Astra Zeneca and is presently used to test GolfIT, the Swedish golf foundations new IT system that is under development by WM-Data.

The part of SIMON that this thesis focus can be described as the step from general flow into a specific test case. All test cases are based on the different functions in the to-be-tested application and were previously manually written into an XML file called a Business Information File, from now on referred to as a BIF.

The desire from WM-Data was for me to examine the process when creating, editing and maintaining the BIFs and formulate a solution that would replace the current manual method; using a simple text editor that offers no protection against misspellings or errors in the syntax and was very time consuming.

The suggestion for a replacing solution to handle the BIFs offered by me would then have to be evaluated and validated to examine if any advantages was gained or not.
1.2 Objectives

The main objective for this thesis was for me to make an own solution to the specified problem regarding the BIFs at WM-Data and then evaluate the solution to find out if it brought with itself any advantages.

To be able to achieve this I divided the whole projects workflow into subparts with different objectives:

- Literature studies about automated testing to gain a greater understanding of the subject.
- Analyze the situation at WM-Data, identify the problems and design a solution.
- Implement and test the solution.
- Evaluate the solution according to relevant variables from the literature to identify advantages or possible disadvantages.
- Draw conclusions and propose new action to be taken.

1.3 Hypothesis

My research assignment at WM-Data was to find a solution that replaced the previous manual method for dealing with the BIFs. The new semi-automated method that I developed - called the BIF-Editor - was intended to simplify the creation of BIFs by ensuring correct syntax, spelling, and structure and make the process faster and easier for the user.

If all of these benefits were realized, could this enable the test cases produced to have higher quality?

So to evaluate if the quality of the BIFs created with the BIF-Editor increased and if it brought other features, like error avoidance and faster creation, with it itself I measured the path cover of the BIFs. This is described in the following hypothesis:

The test cases produced when using the new semi-automated method with the BIF-Editor has higher quality regarding path coverage compared to the previous manual method.

The variable aiming to evaluate the quality of the test case was the path coverage, which is described in the chapter below, but other variables were also taken into account for the evaluation. The speed of the creation and the error ratio both of the manually created BIFs and the semi-automated with the BIF-Editor were also investigated, which is described in the following sub-questions:

- Did the speed of creation increase when using the BIF-Editor?
- Did the error ratio decrease then using the BIF-Editor?
1.3.1 Path cover

There are many different test techniques available with various views and approaches that can reveal different kinds of bugs in an application. The cornerstone of all these techniques is path testing where - according to Boris Beizer - “a path through a routine is any executable sequence of instructions through that routine”. [Beizer 1996]

Since there may be loops in a function (routine) and that every decision doubles the number of paths in the function there may be an infinite number of paths and this is of course unachievable to test. Therefore path testing is conducted with an amount of different paths which, Beizer writes, has the objective “to exercise enough different paths to demonstrate that the routines actual structure matches its intended structure”. [Beizer 1996]

For test cases to provide complete path cover they must, at a minimum, contain enough paths so:

1. Every instruction in the routine has been exercised at least once.
2. Every decision (branch or case statement) has been taken in each possible direction at least once.

Despite its name, complete cover does not however mean complete testing. To test all possible paths is, as mention earlier, unachievable but with complete cover we are assured that every instruction has been exercised at least once and that every alternative has been tried at least once. [Beizer 1996]

Beizers opinion is that it is not possible to test and integrate a system in a proper manner without having complete cover. Dustin on the other hand points out two major drawbacks that instead only makes path cover suited for critical success functions. First, the number of paths can exceed what is possible to cover, be too exhaustive and beyond the scope of most test programs. Second, path covering is very time consuming and there is often limited time. [Beizer 1996] [Dustin 1999]

1.4 Delimitations

The thesis is delimited to analysis, design and implementation in the development process. The results have been drawn according to my chosen method of evaluation and its specified variables.
2 **SOFTWARE TESTING**

For as long as software has been developed there has been testing because bugs are sadly unavoidable and together with shrinking deadlines limiting the time before delivery, some effective form of testing must be performed to be able to assure the quality of the software. [Beizer 1996]

Test phases have therefore always been a part of the software development cycle, but there have however been different views and approaches towards testing along the way. This chapter presents an overview of testing with its history, narrowing in on automated testing with Rational Robot and finally on SIMON. [Marciniak 1994]

2.1 **History of software testing**

The complexity of software has enabled testing since the very first software was written. Test scenarios during this time was written down on paper and performed manually at the very end of the project schedule with thoughts of exhaustive testing of the entire software. And initially a finite set of test procedures could effectively test a complete system, but as the systems grew, so did their complexity. [Marciniak 1994]

New architectures such as client-server emerged and the tests could no longer exercise a single, closed application running on a single computer as in the past. With new aspects, such as systems running on different platforms with its functionality over networks, the way of performing the testing also had to be developed. [Dustin 1999]

So to be able to meet all the new demands, automated test tools were developed with the aim to prevent bugs from occurring. These new tools were more complex then earlier, demanding a greater effort and more planning performed by specialized personnel familiar with the application under test. [Dustin 1999]

Testing was no longer a minor detail in the development of software; it had matured and expanded to be “more of a programming exercise, although if continues to involve the traditional test management functions such as requirements traceability, test planning, test design, and test scenario and script development”, Dustin writes. [Dustin 1999]

Since then, different kinds of automated test tools has been developed, all of them with the aim to minimize project schedule and effort but maintain the maximum quality. [Marciniak 1994]
2.1.1 Purpose and quality of testing

To assure the quality of an application, some form of testing must be performed. Otherwise, you can not guarantee that the application will work in a proper and desirable manner until it actually is implemented and in use, so to evaluate the application and put in under test; test cases can be produced and run to find bugs. [Marciniak 1994]

But since time is limited and the number of possible test cases normally is infinite, the testing must be performed in a very efficient manner. The selection of which parts of the system to focus the tests on and the selection of test cases are important questions since it is this small amount of chosen test cases that is expected to find most of the bugs in the software. [Fewster 1999]

Fewster defines four attributes that describes the quality of a test case:

1. How effective it is to detect defects.
2. How many things it tests; it should test more than one thing to reduce the total number of test cases required.
3. How economical it is to perform, analyze and debug.
4. How much maintenance effort it requires each time the software changes.

Often, these attributes must be balanced against each other to get test cases that find a high proportion of bugs, as well as avoiding excessive cost. [Fewster 1999]

Depending on the objectives and the size of the testing, different methods might be applied and the tests can be directed towards certain aspects. The method of choice previously might very well have been to perform the tests manually, but in order to generate more economic and evolvable tests - and still maintain the same test quality - automating the test could be a better approach. [Fewster 1999]
2.2 Automated testing

Dustin et al define automated testing as: “The management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated test tool”. With an automated testing application you automate the tests, which during the development and integration stages offer significant payback by running the test scripts repeatedly. [Dustin 1999]

When automating tests the quality of the individual test cases remains the same, how effective the tests are regarding detecting defects or testing more than one thing is not affected. But it affects how cost effective and evolvable it is, since once it is implemented the cost of running the automated test is often significantly smaller compared to the manual method. “After the automated test has been run a number of times it will become much more economic than the same test performed manually” Fewster claims and shows this in the following diagram below:

Apart from the economic advantage, Fewster also points out the following benefits of automated testing compared to manual testing:

1. **Run existing tests on a new version of the application:**

   Given that the tests already exist and have been automated to run on an earlier version of the program, it should be possible to select the tests and initiate their execution with just a few minutes of manual effort.

2. **Run more tests more often:**

   The ability to run more tests in less time obviously makes it possible to run them more often.
3. **Perform tests which would be impossible to do manually:**

   To perform for example a live test of an online system with a couple of hundred users would be impossible to perform manually, but with an automated testing application the users can be simulated.

4. **Better use of resources:**

   Having skilled employees manually repeatedly entering the same test inputs over and over again is not a very good use of resources. With an automated test tool the test could instead be performed overnight by a machine.

5. **Consistency and repeatability of tests:**

   Tests that are repeated automatically will be exactly the same every time keeping the consistence on a level that is very difficult to achieve manually.

6. **Reuse of tests:**

   The effort of choosing what to test, designing the test and maintaining it is worth spending time on since it often can be reused.

7. **Earlier time to market:**

   The speed of the automated testing compared to the manual makes the tested application ready for the market earlier.

8. **Increased confidence:**

   The chance of any unpleasant surprises after the system has been released will decrease if extensive automated tests have been run successfully.

   [Fewster 1999]

To ensure successful implementation of automated testing a structured methodology called Automated Test Life-Cycle Methodology (ATLM) can be applied. ATLM is used for designing and executing test activities and supports the decision whether the application is suitable for automated testing or not.

[Dustin 1999]
2.2.1 Testing interfaces

If the application is considered to be suited for automated testing, then there are three different interfaces available for testing:

- Command Line Interface (CLI)
- Application Programming Interface (API)
- Graphical User Interface (GUI)

Some applications have all of these interfaces, while other only have one or two that are available for performing tests on. These different interfaces provide for various kinds of testing, but they all have the same basic meaning; to test the underlying application. The interfaces are entrances to the same application, but they have different approaches and prerequisites towards it. [Zambelich]

A visual description of this is presented below:

Out of these interfaces, API and CLI testing are considered to be easier to automate than GUI testing according to Zambelich. To explain this statement, the following is a description of some of the difficulties with GUI test automation that Bret Pettichord points out, which does not appear when testing through the API or CLI:

- The record/playback feature (explained in detail in the next chapter) in the automation tools, used to record manually performed tests that later can be re-run, rarely works which often forces the tester to primarily test the GUI by hand.

- To get the automated tool to work with your application can often be quite a technical challenge that requires considerable expertise by the tester. For example, non-standard or custom GUI controls can present added difficulties since the automated testing tool might not be familiar with them, which often require modifications to the product source code or patches and updates from the automated tool developer. This difficulty is also one of the reasons why the GUI test tools are so expensive.
• To maintain working automated GUI testing, you have to keep up with the design changes made to the GUI. The GUI is constantly modified and improved throughout the entire development process and since the GUI testing is dependent on the details in the GUI, the testing has to be updated to match the changing interface. [Zambelich] [Pettichord]

The API and CLI on the other hand tend to undergo less changes after the original design has been performed, which most often makes them easier to automate than GUI testing. Obviously, the GUI has to be tested as well, but these are reasons that suggest why the testing of the applications functionality not should depend on the GUI alone. Instead, Pettichord recommends that there should be additional tests performed through the CLI or API to guarantee the quality of the application when the GUI is redesigned. [Pettichord]

Which interface to perform the testing through, I leave unsaid since all applications are different and in different stages in the development cycle. This makes every single situation unique and a thorough investigation is needed to evaluate which method that suits the current application the most. Therefore I will narrow further in on GUI testing since this is what is used by SIMON and is the subject for this thesis.

The automated testing application Rational Robot is initially used in the following chapter to give an overall picture of GUI testing and its weaknesses, which are the origin of SIMON.
2.3 GUI testing

Rational Robot is an example of an automated testing application that is used for functional testing performed through the GUI. When an application under development reaches a state where intensive testing of the underlying functions is suitable to take place, an automated testing application such as Robot can be used to test the code with record/playback tests. [Rational]

2.3.1 Record/playback test system

With record/playback tests, test scripts that can re-run a certain scenario are recorded to test one or more functions in an application. This is done via executing the test manually on the application under test while the test tool records every mouse movement, click and coordinates in the GUI. With this approach test scripts, i.e. test cases, can be generated that afterwards can be re-run and further test the certain scenario. [Rational]

The main advantages with these kinds of testing applications, like Robot, that uses record/playback testing is, as mentioned earlier, the economic aspect when compared to the manual method. But there are however several drawbacks with this kind of tests, such as these Keith Zambelich points out:

- The test scripts resulting from this method contain hard-coded values (a buttons name in the GUI for example) which must change if anything at all changes in the application. This means that if details in the GUI changes, the tests have to be re-recorded.

- The costs associated with maintaining such scripts are astronomical, and unacceptable. After all, the fundamental thought with record/playback test scripts is that a recorded scenario should be run over and over again to gain economic advantages, but these are lost if the scripts have to be re-recorded every time changes are made in the GUI.

- These scripts are not reliable, even if the application has not changed, and often fail on replay (pop-up windows, messages, and other things can happen that did not happen when the test was recorded).

- If the tester makes an error entering data, etc., the test must be re-recorded.

- All that is being tested are things that already work. Areas that have errors are encountered in the recording process (which is manual testing, after all). These bugs are reported, but a script cannot be recorded until the software is corrected. So what are you testing? [Zambelich]

It is these limitations in testing applications such as Rational Robot, mostly concerning the hard-coded GUI details, which are the underlying reasons for the development of WM-Datas own automated testing system SIMON.
2.4 SIMON

As mentioned, SIMON emerged from limitations of the widely used testing application Robot whose test scripts have to be re-recorded to work properly again when changes in the GUI are made. With SIMON on the other hand, the test scripts does not contain any GUI details so even if, for example, buttons are moved or if their names change, the test scripts will still be functional. How does this work?

This is possible because, the whole idea with SIMON is several different levels of abstraction and the GUI details specified in a different level than in Robot. Instead of directly basing the test scripts on the GUI details like Robot does, SIMON does this indirectly from its test scripts; the BIFs.

The information of the GUI details can be found in a separate level called the Business Information Logic (BIL) model. So instead of directly calling the GUI from the test scripts they are called through the BIL model. This feature means that any changes to the GUI during the development of the application only have to be updated in one single place; in the BIL model.

Another one of SIMONs features is that when a test has been run it generates report documents of the outcome where it gathers screen dumps, results and explanations about the performed test. Robot on the other hand basically just presents if the test succeeded or not, not offering any explanations or summary of the outcome.

2.4.1 Business Information Logic

The underlying logic behind the application under test is kept in a model called the BIL model and this specifies how SIMON should “consume” the BIFs. Every function in the application that should be possible to test with SIMON has to be described in the BIL model, regarding all the GUI details in every state of the application. This instructs SIMON about how to navigate through the GUI, meaning where and when to click and so on, to perform the requested function.

As mentioned earlier, the BIFs are indirectly using the GUI to test functions and this means that when a BIF tests a certain function in the application, this is performed through the BIL model which have the GUI details for the entire functions scenario specified. In this manner all GUI details are kept out of the test scripts and updates only have to be performed in the BIL model, not in every single test script to make them work properly again. How this works in practice is further described in the next chapter where examples of a BIF calling the BIL model are shown.
2.4.2 Business Information File

A Business Information File (BIF) is an XML file with tags containing instructions for SIMON to perform when testing an application, with other words; a BIF is a test case or a test script for that matter. A BIF can consist of two different kinds of instructions, either stimuli or responses.

A stimuli is an instruction that tells SIMON to perform an action, use a certain function in the tested system. The following is an example of a stimuli instruction telling SIMON to perform an action on the application under test called “nytt_uppdrag”:

```xml
<stimuli name="nytt_uppdrag"/>
```

Exactly what SIMON should perform when a stimuli like “nytt_uppdrag” above is called is defined in the BIL model regarding where and when to click in the GUI. The stimuli tell SIMON to perform the instructions specified in the BIL model that describes the function in the tested application. This enables the function in the application, which sets up certain screen scenarios with different input fields depending on the current state of the application. These different screens with various input fields require input-data from the user, in this case from the BIF, and these can be found in the response tags.

The following response tells SIMON which data to use in the “login_information” state of the application, which is the first state of the “nytt_uppdrag” function:

```xml
<response query="login_information">
   <group>
      <field name="site">192.168.0.1</field>
      <field name="org_enhet">Lidhems GK</field>
   </group>
</response>
```

(These rows were copied from a whole BIF that can be found in appendix 9.3)

The “login_information” response above holds the input data for this state which has one input field for “site” and one for “org_enhet”. And when this response has been performed by SIMON the next state, all according to the underlying flowchart as described in the next chapter, will be processed and run.

These BIFs was previously created and updated manually using editors like Notepad which did not offer any protection or help against misspellings or errors in the structure. Another disadvantage with this old approach was that it demanded expert knowledge within the domain of SIMON to be able to create a BIF and took a long time to produce, even for the experts and this was the reason for my project.

Thomas Pihl at WM-Datas idea was for me to come up with a solution that would replace the old manual method with some kind of semi-automate to handle the creation and handling of the BIFs instead.
2.4.2.1 Flowchart

All scenarios, functions in the application under test, which are tested in the BIFs, descend from flowcharts that describe their structure, states, names and valid paths. This is often described using the modeling language UML, which graphically using boxes and lines gives the viewer an overview of the function. An example of a flowchart describing the function “nytt uppdrag” in the WM-Data developed software GolfIT, which SIMON is used to test, looks like this:
2.4.3 BIF-Editor

My solution for easier creation and maintaining of the BIFs is an XML-based Java application called the BIF-Editor. The BIF-Editor was developed under six to seven weeks after initial analysis and preparations with the aim to enable efficient transformation from general flow into a specific test case by making the process faster and guaranteeing correct structure and syntax.

Since a BIF is a test case based on a flowchart describing a certain function, the initial step for creating BIFs is to translate the flowcharts into “rulesets” in XML. These rulesets are essentially just another way of describing the flowcharts, but in a way the BIF-Editor can interpret. They do however also contain additional information necessary for the creation such as the input fields names but most importantly; they make sure that the correct structure and syntax is used.

With the BIF-Editor, the user can choose to create a new BIF - or edit/update an existing BIF - and design it according to his own choice, but as mentioned it is always based on the underlying ruleset that guarantee correct structure and syntax.

Following is a couple of screenshots explaining the process when a BIF is created using the BIF-Editor:

1. After starting the BIF-Editor, the initial step is to choose a ruleset; the kind of BIF we want to create:
2. When a ruleset has been chosen the BIF-Editor parses it and sets up the application according to the structure and syntax in the ruleset:

3. The user can now create a BIF from his choice by filling in the text fields and navigating through the ruleset in a structured, but yet free manner:
4. When done, the user can save the BIF and a new test case has been created:

(An example of the underlying flowchart, its translation into a ruleset and a generated BIF can be found in appendix 9.1, 9.2 and 9.3)
3 EXTENSIBLE MARKUP LANGUAGE

SIMON is an XML-based automated testing application and as you have seen the BIFs and rulesets are written in XML. XML is short for eXtensible Markup Language and has its origin in the old markup languages SGML and HTML.

3.1 Markup Languages

In 1969 IBM Research invented the first modern markup language, a language used to describe new languages with, called Generalized Markup Language (GML) which later became Standard Generalized Markup Language (SGML). In 1986, SGML became international data storage and exchange standard as ISO 8879 and was widely used by, for example, the U.S. government because of its features as an extremely powerful markup language. [Anderson, et al 2000]

With SGML documents could be shared between multiple users on different platforms with retained information about the documents structure. But SGML was designed by engineers, for engineers and was rather complicated with its extensive standard, and with the advances in the development of Internet, a simpler approach was needed. [Pfaffenberger 1999] [Anderson, et al 2000]

3.2 HTML

Hypertext Markup Language (HTML) was introduced as a simple and platform independent language with the aim to make information available all over the world on Internet. In 1990, HTML became a World Wide Web Consortium (W3C) recommendation and is by far the most known and used language on Internet. [Statskontoret]

With HTML there only exists a certain amount of predefined markup tags, like for example <b> for bold </b>, that can be used, so HTML is what we call a static language; it can not be changed interactively. In the beginning, when Internet sites merely contained some general information about a company or about a person then HTML was suitable, but as the use of Internet grew so did the demand for more advanced applications. [Statskontoret]

HTML, with its feature of only describing how the document should look like and not the content in the document, together with the restricted amount of tags did not meet the requirements of the new era of applications for Internet. So in 1996 W3C began the process of creating XML, a new markup language that would have the flexibility and power of SGML but also the simplicity and widespread acceptance of HTML. [Statskontoret] [Anderson, et al 2000]
### 3.3 XML

XML was initially developed by scientists and experts from various fields within the domain of computer science and technology in the SGML Working Group gathered by W3C. Their first draft was published late 1996, but it was not until February 1998 that XML became a W3C recommendation. [Anderson, et al 2000]

As Anderson, et al mentions, XML has been described as “the ASCII of the Web” with its simple and standardized way to delimit text data and is a great way to share information over Internet because:

- It is an open standard; data exchange over different platforms can be performed in an independent manner.
- Self-describing; readable for the human eye and easy to understand.
- Separates data from design; XML only describes the data and not the layout which makes it possible to present the data in various medias.
- Data sharing; no prior coordination has to be done to be able to share data between programs.
- One internet standard; with XML only one single standard exists to avoid multiple different standards as with HTML. [Anderson, et al 2000]

The fundamental of XML is that it is a method to store information in a flexible and simple manner. It is an extensible markup language since it does not have any predefined tags, the users defines them himself depending on his needs. [Anderson, et al 2000]
4 THE CREATION OF THE BIF-EDITOR

The practical, as well as the underlying, part of this thesis was the creation of the BIF-Editor that made out the foundation of the evaluation and the results drawn.

4.1 Analyze, design and implementation

There was obviously quite a lot of information about how SIMON works, its architecture and development, that I initially had to learn to be able to perform a good analyze of the situation. My supervisor at WM-Data helped me from the start to get going and after several old BIFs had been analyzed I gained the necessary understanding to start designing an own solution to replace the manual creation. The design evolved and changes were made over time, which eventually resulted in the following design that is basically described below:

![Diagram of BIF-Editor design]

This design was then implemented using Java 1.4.1 with Swing to build the GUI. The XML files; the BIFs and rulesets was parsed and managed as Document Object Models (DOM) and was traversed using XPath.
5 EVALUATING THE BIF-EDITOR

To validate my hypothesis, my own solution of the specified problem at WM-Data, the BIF-Editor had to be evaluated and form the results. The most suiting method of research to accomplish this was action research, which is described by Herbert in Dawsons book as “the carefully documented (and monitored) study of an attempt by you … to actively solve a problem and/or change a situation”. [Dawson 2000]

The aspects this evaluation focused on were the quality of the BIFs produced regarding their path cover, the speed of creation and their error ratio. In addition to these variables in the evaluation, interviews/discussions and observations was also performed, not so much to answer the hypothesis but to get spontaneous opinions and ideas from the employees testing the BIF-Editor.

5.1 Action research

For this industry based/problem solving project, action research was used as research method which according to Kurt Lewin is “a three-step spiral process of (1) planning which involves reconnaissance; (2) taking actions; and (3) fact-finding about the results of the action”. [Lewin 1946]

Basically with action research, you formulate general hypothesis concerning the identified problems and specify appropriate actions that will lead to improvements (hopefully) of the situation. These actions are then taken, and to see the effects the situation is evaluated to audit the earlier formulated hypothesis. [Bell 2000]

New hypothesis are then drawn based on the preceding evaluation and new actions can be specified to solve the current problems. In this spiral manner you can continue to gain greater understanding and improvements in practice. [Bell 2000]

With this thesis the initial circle in the spiral process has been performed with the creation of the BIF-Editor, which was the first step in the improvement of producing higher quality BIFs. The development of the BIF-Editor will continue after the end of this project with this evaluation as foundation for the appropriate new actions to be taken in order to improve it further.
5.2 Evaluating path cover

To measure the path cover I reduced the flowchart that the evaluation was based on into a graph and examined its nodes and link coverage in a table according to Boris Beizer.

An example of a flowchart described as a graph:

![Flowchart Diagram]

The table specifying the paths:

<table>
<thead>
<tr>
<th>PATHS</th>
<th>DECISIONS</th>
<th>PROCESS-LINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>aabcdef</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>abdghedc</td>
<td>NO, YES</td>
<td>NO, YES</td>
</tr>
<tr>
<td>abdefgade</td>
<td>YES, YES</td>
<td>YES, YES</td>
</tr>
<tr>
<td>abcdffhede</td>
<td>YES, NO</td>
<td>YES, NO</td>
</tr>
</tbody>
</table>

The tactics regarding the paths are:

1. A sufficient number of paths to achieve coverage.
2. Selection of short, functionally sensible paths.
3. Minimizing the number of changes from path to path – preferably only one decision changing at a time.
4. Favor more but simpler paths over fewer, complicated paths.

[Beizer 1996]

5.2.1.1 Speed of creation and error ratio

Together with the evaluation of the path cover, the two additional variables; speed of creation and error ratio was also measured. The speed of creation was simply measured by timing the creation of BIFs, both when produced manually and with the BIF-Editor. The error ratios of these BIFs were then achieved by examining the structure and syntax. The structure was checked against the flowchart that describes the correct flow and to validate the syntax the BIFs were checked in Internet Explorer.
5.2.2 Interviews/discussions and observations

During the evaluation sessions interviews/discussions were also held with the employees creating the BIFs to get his/hers opinions on the BIF-Editor. But it was also used for me to gain greater understanding of how other people interpret the situation and perceive the BIF-Editor.

The interviews were steered/focused interviews which meant that I did not have a form with questions for the interviewee to answer. The interviews were more loosely structured, but with different predefined themes to cover and with space for open discussions and thoughts. [Bell 2000] [Kylén 1993]

This was combined with participating observations when the application was tested to get the employees spontaneous reactions and opinions. These observations functioned as a complement to the interviews to provide for more evaluation through unpredicted thoughts and approaches. [Bell 2000] [Kylén 1993]

These interviews/discussion and observations was mostly used for me as the developer of the BIF-Editor to get an evaluation of the application. They were not primarily used for the evaluation of the quality of the BIFs, but simply as a complement that seemed as a natural and obvious thing to perform at the time with the aim to develop new thoughts and perspectives.

This data was then mostly used for the discussion, suggestions to further studies and for the possible next round of actions to be taken, all according to the research method of action research.
5.3 Conducting the evaluation

When the BIF-Editor was finished the evaluation was conducted. This was performed after the underlying flowchart had been translated into a ruleset and the employees had been given instructions about how to operate the BIF-Editor.

5.3.1 Participating profiles

All evaluation was performed using employees at WM-Data that has been working with the development of SIMON since its first version. They have previously been the ones manually writing all the BIFs for the earlier applications under test and are experts on constructing test cases for this particular automated testing application.

5.3.2 Planning and preparations

To be able to evaluate anything at all, the fundamental step when conducting this thesis was to design and implement the BIF-Editor. Since this was to be the foundation for the evaluation of this whole project it was obviously very important to get a useful application to work.

So when a functional version of the BIF-Editor was finished an evaluation according to speed, error ratio and path cover was carried out by comparing the quality of the manually created BIFs with the semi-automated BIFs using the BIF-Editor.

Since the later part of the evaluation was performed with the BIF-Editor, the flowchart functioning as foundation for the evaluation was translated by me into an XML ruleset. Additional preparations also included giving the employees, which performed the evaluation later, basic instructions about how to use the BIF-Editor (and this was obviously not performed using the same ruleset that was later used during the evaluation).

5.3.3 The evaluation procedure

The evaluation consisted of two parts; the first part was to let an employee produce BIFs manually in the same manner as he/she had done earlier. That gave me BIFs to measure the quality on regarding the path coverage, but it also gave me an estimated time of long time the creation took and how high the error ratio were when the BIFs created were examined and tested.

The second part of the evaluation was to let the same employee use the BIF-Editor to produce BIFs during the same time that elapsed during the first part. This gave me BIFs to compare with the previously manually created BIFs, which enabled me to measure if these BIFs had higher path cover or not. But it also gave me data about whether the creation took longer or shorter time than earlier and if the error ratio increased or decreased when using the BIF-Editor.
This evaluation was performed twice by two different employees, but in reversed order the second time using the BIF-Editor first and the manual method afterwards. This was performed to further strengthen the evaluation and to see if the order of creation mattered.

5.3.4 The flowchart described as a graph

Both rounds of the evaluation were performed using the same underlying flowchart, “registrera bana”, which can be found in appendix 9.4. This flowchart is described below in the following graph, all according to Beizer, and was used to form the results regarding the path cover:

5.3.4.1 Explanation to the graph

1. start
2. slut
3. login
4. ny_bana
5. välj_slinga
6. ny_starttids-regel
7. ny_upplysning
8. ny_starttidsförbud
9. ny_reservation
10. ny_hcp-regel
6 RESULT OF EVALUATION

The results from the evaluation are presented in the same manner as it was carried out; two rounds of evaluation with the order reversed the second time. These two rounds gave me sub conclusions with data regarding the path coverage, the time of creation and error ratio that were summarized and made the final results.

6.1 First round of evaluation

The old manual method was first used followed by the usage of the BIF-Editor during the same amount of time that elapsed for the employee to manually produce the first BIF.

6.1.1 Manual creation of BIFs

<table>
<thead>
<tr>
<th>Paths</th>
<th>Decisions</th>
<th>Process-link (instruction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c d e f g h i j k l m n o p</td>
<td>5 6 7 8 9 10</td>
<td>x x x x x x x x x x x x x</td>
</tr>
</tbody>
</table>

With the manual method it took the employee 27 minutes to create just one BIF and a couple of extra minutes to correct the two syntax errors that was in the created BIF, which added up to a total of 29-30 minutes.

6.1.2 Semi-automated creation of BIFs using the BIF-Editor

<table>
<thead>
<tr>
<th>Paths</th>
<th>Decisions</th>
<th>Process-link (instruction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c d e f g h i j k l m n o p</td>
<td>5 6 7 8 9 10</td>
<td>x x x x x x x x x x x x x</td>
</tr>
</tbody>
</table>

The second part of the first round of evaluation was performed by the same employee using the BIF-Editor during barely 30 minutes, which enabled the employee to produce three BIFs that did not contain any errors when checked.
## 6.1.2 Summary of the first round of evaluation

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Error ratio</th>
<th>Path cover</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manually</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIF 1</td>
<td>27 + 2 minutes</td>
<td>2 errors</td>
<td>7/12 decisions, 10/16 instructions</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>29 minutes</td>
<td>2 errors</td>
<td>7/12 decisions 10/16 instructions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BIF-Editor</strong></th>
<th>Time</th>
<th>Error ratio</th>
<th>Path cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIF 1</td>
<td>10 minutes</td>
<td>0 errors</td>
<td>8/12 decisions, 12/16 instructions</td>
</tr>
<tr>
<td>BIF 2</td>
<td>10 minutes</td>
<td>0 errors</td>
<td>10/12 decisions, 13/16 instructions</td>
</tr>
<tr>
<td>BIF 3</td>
<td>8 minutes</td>
<td>0 errors</td>
<td>9/12 decisions, 12/16 instructions</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>28 minutes</td>
<td>0 errors</td>
<td>12/12 decisions 16/16 instructions</td>
</tr>
</tbody>
</table>
6.2 Second round of evaluation

The second round of evaluation was performed by another employee, with the same experience of creating BIFs, but this time the BIF-Editor was used first to produce BIFs before the old manual method was used during the same elapsed time.

6.2.1.1 Semi-automated creation of BIFs using the BIF-Editor

<table>
<thead>
<tr>
<th>Paths</th>
<th>Decisions</th>
<th>Process-link (instruction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c e g i m k n o p d e f h i j k l m n o p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6 7 8 9 10</td>
<td>a b c d e f g h i j k l m n o p</td>
</tr>
<tr>
<td>abcefgikmn</td>
<td>Y N Y Y N Y</td>
<td>x x x x x x x x x x</td>
</tr>
<tr>
<td>nopcdde</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abceghhhhi</td>
<td>Y Y N Y N Y</td>
<td>x x x x x x x x x x</td>
</tr>
<tr>
<td>mnn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>abceffghhikl</td>
<td>Y N Y Y Y Y</td>
<td>x x x x x x x x x x</td>
</tr>
<tr>
<td>llm0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The first part of the second round of evaluation performed with the BIF-Editor resulted in three BIFs created in 28 minutes with no errors.

6.2.1.2 Manual creation of BIFs

<table>
<thead>
<tr>
<th>Paths</th>
<th>Decisions</th>
<th>Process-link (instruction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c e g i m o p d e f h i j k l m n o p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6 7 8 9 10</td>
<td>a b c d e f g h i j k l m n o p</td>
</tr>
<tr>
<td>abcefgikmo</td>
<td>Y Y Y Y Y Y</td>
<td>x x x x x x x x x x</td>
</tr>
<tr>
<td>efghkmo</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With 28 minutes available, one BIF were afterwards manually created by the same employee. Similar to the BIFs created with the BIF-Editor, no errors did occur.

6.2.2 Summary of the second round of evaluation

<table>
<thead>
<tr>
<th>BIF-Editor</th>
<th>Time</th>
<th>Error ratio</th>
<th>Path cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIF 4</td>
<td>10 minutes</td>
<td>0 errors</td>
<td>9/12 decisions, 13/16 instructions</td>
</tr>
<tr>
<td>BIF 5</td>
<td>9 minutes</td>
<td>0 errors</td>
<td>8/12 decisions, 11/16 instructions</td>
</tr>
<tr>
<td>BIF 6</td>
<td>9 minutes</td>
<td>0 errors</td>
<td>8/12 decisions, 12/16 instructions</td>
</tr>
<tr>
<td>Summary</td>
<td>28 minutes</td>
<td>0 errors</td>
<td>11/12 decisions 15/16 instructions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Manually</th>
<th>Time</th>
<th>Error ratio</th>
<th>Path cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIF 2</td>
<td>27 minutes</td>
<td>0 errors</td>
<td>8/12 decisions, 12/16 instructions</td>
</tr>
<tr>
<td>Summary</td>
<td>27 minutes</td>
<td>0 errors</td>
<td>8/12 decisions 12/16 instructions</td>
</tr>
</tbody>
</table>
6.3 Summary of evaluation

The three BIFs created using the BIF-Editor during the first round of evaluation did achieve complete cover; all decisions were taken in each direction and every instruction were exercised at least once. In the second round when the BIF-Editor was used, complete cover was almost achieved; 11/12 decisions were taken and 15/16 instructions were exercised.

With the manual method, only one BIF each time was possible to be produced during the same time frame as when using the BIF-Editor. This did not enable the employees to reach the same qualities regarding path cover; 7/12 decisions were taken and 10/16 instructions were processed on the first time of evaluation and 8/12 decisions and 12/16 instructions the second time.

There also occurred two syntax errors when producing BIFs manually, but none when using the BIF-Editor. These figures gave me positive answers to my sub-questions:

- Did the speed of creation increase when using the BIF-Editor?
- Did the error ratio decrease then using the BIF-Editor?

The answers to both of these questions are yes and this together with the previous presented results regarding the path cover means that the error ratio decreased, the speed of creation rate increased and therefore the quality regarding path coverage increased. This proves my hypothesis and supports that:

The test cases produced when using the new semi-automated method with the BIF-Editor has higher quality regarding path coverage compared to the previous manual method.
7 DISCUSSION

Following are my own thoughts and ideas that arouse during the writing of this thesis, but also questions that emerged during the interviews/discussions held with the employees during the evaluation. But first, the validity and reliability of this thesis is discussed below.

7.1 Validity and reliability of this thesis

The usage of the BIF-Editor resulted in BIFs of higher quality regarding the path cover, but was this really achieved by the BIF-Editor? It probably was not.

The increased path coverage was probably on the other hand achieved because of the simple fact that the BIF-Editor enabled the employee to produce more BIFs than previously possible during the same time. The individual quality of the BIFs, created manually or not, was unchanged, but with an increased speed of creation using the BIF-Editor more BIFs could be created which raised the path coverage.

This statement has obviously not been proved, but since the BIF-Editor in the current version has no support or feature that increases the quality automatically then this would be a reasonable thought. It could even be so that when using the BIF-Editor the creation of BIFs becomes less thought through than previously and the quality of the individual BIFs might actually decrease.

Further investigations would obviously have to be performed to evaluate this, since this thesis merely introduced the usage of the BIF-Editor and this evaluation had a different alignment.

As mentioned last in chapter 5.3.3, the second round of evaluation was performed in reversed order to see if the order of creation mattered. But from analyzing the results afterwards no clear conclusions can be drawn. The individual quality of the BIFs was about the same, whether they had been created manually or not or if the BIF-Editor was used before or after the manual method. If this was coincidence or if it proves a fair evaluation, I will not speculate about.

The error ratios of the manually created BIFs were two in the first round and zero in the second. These were fairly low ratios, but there are however significant chances that errors are done when manually dealing with larger, more complex flows. The results of this evaluation probably show a lower error ratio than what the actually overall ratio really is.

Each one of the BIFs created during the evaluation was about four pages long when printed out so I really consider the chance of making so few errors as was done during this evaluation as highly unlikely. If this was pure luck or if the employees have gotten so good to create the BIFs, I leave unsaid.
7.2 Future studies

The interviews/discussions with the employees during the evaluation gave some new ideas for a possible next round in the spiral process of action research. As mentioned when describing action research earlier, new actions are taken based on the results from the previous evaluation and these are then performed and later also evaluated.

The basic idea that was discussed during the evaluation was to deploy the creation of the BIFs using the BIF-Editor to the intended users since they are the ones having the real expert knowledge about the application under test. They are the ones who have worked with the old system and they know the terminology, so with the use of the BIF-Editor the deployment would hopefully not be too difficult.

As an action researcher, if I were to do the next circle within this action research manner I could now investigate and evaluate if the intended users of the system could produce BIFs with a preserved, or hopefully higher, quality. This would obviously also involve evaluating other aspects of the BIF-Editor such as its usability and correctness etc. Perhaps the intended users do not have the demanded computer competence needed which would make this deployment impossible, but if not so, could they overtake the creation of the BIFs?

Another thought that came up during the development of the BIF-Editor was; if a function that out of a flowchart automatically could generate test cases that provided complete path coverage would be a useful feature? The basic thought would then be to let this feature out of a flowchart, or its ruleset, build “BIF skeletons” that automatically provided complete cover which would guarantee the quality regarding the path coverage. Maybe these suggestions could be subjects for further studies within the field of automated testing and SIMON?
8 LITERATURE

8.1 Books


8.2 Web sources


9   APPENDIX

9.1   Flowchart

Nytt uppdrag

Start

login_information

OK

Cancel

uppdrag

OK

Cancel

N/A

addera_deltagare

OK

Cancel

N/A

addera_roll

OK

Cancel

OK

Slut
9.2 Ruleset

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE bil SYSTEM "BusinessInformationLogic.dtd">
<bil>
  <flow name="nytt_uppdrag" version="1.6" modified="2003-03-24">
    <response name="login_information" initial="true">
      <followed_by condition="OK">
        action
      </followed_by>
    </response>
    <response name="action" restart="true">
      <followed_by condition="OK">
        uppdrag
      </followed_by>
    </response>
    <response name="uppdrag" mandatory_qualid="true">
      <followed_by condition="OK">
        addera_deltagare
      </followed_by>
    </response>
    <response name="addera_deltagare" mandatory_qualid="true">
      <followed_by condition="OK">
        addera_roll
      </followed_by>
      <followed_by condition="NA">
        uppdrag
      </followed_by>
    </response>
    <response name="addera_roll">
      <followed_by condition="OK">
        RESTART
      </followed_by>
      <followed_by condition="NA">
        addera_deltagare
      </followed_by>
    </response>
  </flow>
</bil>

<responselist name="nytt_uppdrag">
  <responsedef name="login_information">
    <BGRef termName="site"/>
    <BGRef termName="org_enhet"/>
  </responsedef>
  <responsedef name="action">
    <stimuli name="nytt_uppdrag"/>
  </responsedef>
  <responsedef name="uppdrag">
    <BGRef termName="uppdragsnamn"/>
    <BGRef termName="uppdragsbeskrivning"/>
    <BGRef termName="uppdragstyp"/>
    <BGRef termName="startdatum"/>
    <BGRef termName="slutdatum"/>
  </responsedef>
  <responsedef name="addera_deltagare">
    <qualref name="uppdrag"/>
    <BGRef termName="deltagartyp"/>
  </responsedef>
</responselist>
<responderdef name="addera_roll">
  <qualref name="addera_deltagare"/>
  <BGRef termName="rollnamn"/>
</responderdef>

</responsetlist>
9.3 BIF

<?xml version="1.0" encoding="ISO-8859-1"?>
<testcase-document version="">
  <scenario case="Default" no="1" name="GolfIT" version="0.1"
  testtype="predefined_testcase">
    <ruleset name="nytt_uppdrag" version="1.6" modified="2003-03-24"/>
    <purpose> Skapa ett nytt uppdrag</purpose>
    <comment> Gjort för testning</comment>
    <step name="default">
      <response query="login_information">
        <group>
          <field name="site">192.168.0.1</field>
          <field name="org_enhet">Lidhems GK</field>
        </group>
        </response>
        <response query="action">
          <group>
            <stimuli name="nytt_uppdrag"/>
          </group>
        </response>
        <response query="uppdrag" qualid="A">
          <group>
            <field name="uppdragsnamn">Kalle</field>
            <field name="uppdragsbeskrivning">Bygg green</field>
            <field name="uppdragstyp">Omgående</field>
            <field name="startdatum">2003-04-01</field>
            <field name="slutdatum">2003-05-01</field>
          </group>
        </response>
        <response query="addera_deltagare" qualid="B" qualref="A">
          <group>
            <field name="deltagartyp">Person</field>
            <field name="förnamn">Klabbe</field>
            <field name="efternamn">Kanel</field>
          </group>
        </response>
        <response query="addera_roll" qualref="B">
          <group>
            <field name="rollnamn">Ansvarig</field>
          </group>
        </response>
      </step>
    </scenario>
  </testcase-document>
9.4 Registrera bana

Registrera bana

Start

login

OK

ny_bana

OK

reg_fler

välj_linga

OK

reg_fler

ny_starttid

reg_fler

ok

ny_upplysning

OK

reg_fler

ny_starttid

förlade

OK

reg_fler

ny_reservation

OK

reg_fler

ny_hop-regel

OK

Stat