ON THE USE OF BASE CHOICE STRATEGY FOR TESTING INDUSTRIAL CONTROL SOFTWARE

Simon Eklund
sed13001@student.mdh.se

Examiner: Daniel Sundmark
Mälardalen University, Västerås, Sweden

Supervisor: Eduard Paul Enoiu
Mälardalen University, Västerås, Sweden

May 25, 2016
Abstract

Testing is one of the most important parts of software development. It is used to ensure that the software is of a certain quality. In many situations it is a time consuming task that is manually performed and error prone. In the last couple of years a wide range of techniques for automated test generation have been explored with the goal of performing more efficient testing both in terms of cost and time. Many of these techniques are using combinatorial methods to explore different combinations of test inputs. Base Choice (BC) is a combinatorial method that has been shown to be efficient and effective at detecting faults. However, it is not very well studied how BC compares to manual testing performed by industrial engineers with experience in software testing.

This thesis presents the results of a case study comparing BC testing with manual testing. We investigate the quality of manual tests and tests created using BC strategy in terms of decision coverage, fault detection capability and cost efficiency (in terms of number of test cases). We used recently developed industrial programs written in the IEC 61131-3 FBD language, a popular programming language for embedded software using programmable logic controllers. For generating tests using BC we used the Combinatorial Testing Tool (CTT) developed at Mälardalen University.

The results of this thesis show that manual tests performed significantly better than BC generated tests in terms of achieved decision coverage and fault detection. In average manually written tests achieved 97.38% decision coverage while BC tests suites only achieved 83.10% decision coverage. In fault detection capabilities, manual test suites found in average 88.90% of injected faults compared to 69.53% fault detection by BC generated test suites. We also found that manual tests are slightly shorter in terms of number of tests compared to BC testing. We found that the use of BC is heavily affected by the choice of the base values chosen by the tester. By using more precise base choice values in BC testing may have yielded different results in terms of decision coverage and fault detection.
# Table of Contents

1 Introduction ........................................... 5

2 Background ........................................... 6  
   2.1 Programmable Logic Controllers .................... 6  
   2.2 Combinatorial Testing ............................... 8  
   2.3 Base Choice Criteria ............................... 8

3 Methodology ........................................... 10  
   3.1 Problem Formulation ................................. 10  
   3.2 Case Study Process .................................. 10  
      3.2.1 Selection of Programs ......................... 11  
      3.2.2 Combinatorial Testing Tool .................... 12  
      3.2.3 Test Generation ................................. 12  
      3.2.4 Metrics ........................................ 13  
      3.2.5 Descriptive Statistics ......................... 15  
   3.3 Execution Framework ............................... 15

4 Results and Analysis .................................... 16  
   4.1 Decision Coverage .................................. 16  
   4.2 Fault Detection .................................... 18  
   4.3 Number of Test Cases ............................... 19

5 Discussion and Validity Threats ......................... 21  
   5.1 Discussion .......................................... 21  
   5.2 Threats to Validity ................................ 22

6 Related Work .......................................... 23

7 Conclusion ............................................ 24

8 Future Work ........................................... 25  
   Bibliography .......................................... 27
List of Figures

2.1 PLC hardware. .................................................. 7
2.2 An example of program running on a PLC. .................. 7

3.1 Research methodology. ....................................... 11
3.2 Experiment methodology. .................................... 12
3.3 Functionality of CTT. ........................................ 13

4.1 Decision coverage achieved by each testing approach. .... 17
4.2 Decision coverage comparison. ............................... 18
4.3 Mutation score achieved by each testing approach. ....... 19
4.4 Mutation score comparison. .................................. 19
4.5 Number of test cases created for each testing approach. .. 20
4.6 Number of test cases comparison ............................ 20
List of Tables

2.1 Example of test executed on a PLC program ........................................ 8
2.2 Example of generated test using BC .................................................... 8
3.1 Supported variable types in CTT. ......................................................... 12
3.2 Generation time in milliseconds using CTT. ......................................... 14
3.3 Number of mutants created for each program used in the experiment. ........ 15
4.1 Experiment Results. .............................................................................. 16
5.1 Test Suite for BC using the base choice provided by the test engineer. .... 22
5.2 Test Suite for BC using the base choice selected by the author of this thesis. 22
5.3 Comparison between two BC test suites each with different base values. .... 22
Chapter 1

Introduction

Software development contains several phases which are important to ensure the quality of the program. One of the most important parts is the testing of the software to make sure that the program works as intended. Testing can consume much time for the developers and therefore it is of interest to make this phase as efficient as possible. There are industrial companies where manual testing is a popular method of testing software, such as Bombardier Transportation AB in Sweden. This company is testing software on a regular basis for their Train Control Management System (TCMS) which runs on Programmable Logic Controllers (PLC). However, there are several methods for automatic test generation which can be more cost efficient such as combinatorial testing strategies. Some of these techniques include: All Combination (AC), Each Choice (EC), Orthogonal Array (OA), Base Choice (BC), and Advanced Combinatorial Testing System (ACTS). Several recent empirical studies [1, 2] showed the value of BC in fault detection capabilities, cost efficiency in terms of test cases used but also decision coverage to some extent. Therefore, we were interested to investigate the use of base choice strategy for testing industrial PLC software in comparison with manually written tests. To our knowledge, no studies have been conducted regarding the comparison between BC and manual testing.

In this thesis, results from an empirical case study are presented. The quality of the testing strategies will be measured in terms of decision coverage, fault detection capabilities and cost (in terms of test cases used). Fault detection was measured using mutation analysis which means that faults were created and injected into the programs. Besides manual and BC tests, tests generated using a Random strategy was also used as a baseline. The number of test cases used for Random was equal to the amount used for BC tests. The results were then analyzed using descriptive statistics. All programs used were written in FBD which is a IEC 61131-3 [3] language for PLC programs. For the generation of tests used in the thesis, a tool developed at Mälardalen University was used. This tool was chosen due to its support for BC and Random test generation and the capability to use this tool with PLC programs in their native format.

The results of this thesis showed that manually written tests performed far better than tests created using BC. In decision coverage and fault detection, manual tests were significantly better than both Random and BC tests, while the number of test cases used for all tests were similar in average (12.84 test cases for manual tests and 15.13 test cases for BC and Random). The difference in number of faults found was found to be in average approximately 19% for manual test and BC test. There are several factors which could have influenced the thesis results that might explain the cause for the low fault detection and coverage criteria for BC.

In Chapter 2 we present an overview of PLC hardware and software, combinatorial testing and the base choice criteria. Chapter 3 describes the methodology used throughout the research. Chapter 4 presents the results and the analysis of the conducted experiment in terms of decision coverage, fault detection and number of tests. In Chapter 5 we discuss the results in order to try and explain the yielded results, clarify the limitations and describe the validity threats. Chapter 6 describes related work. Chapter 7 we present our conclusions. In Chapter 8 we suggest possible future research and work.
Chapter 2

Background

Software testing is one of the most important aspects of software development to ensure that software works as intended is crucial in certain application domains, for example in the railway transportation. The software needs to be able to control different train functions. Programmable Logic Controllers (PLC) which are real-time controllers used in numerous industrial domains, are intensely used in the train domain. Software faults can be catastrophic, especially in railway transportation, and therefore the software is developed to follow certain safety standards (i.e., according to CENELEC 50128 [4]). These standards are mandating that testing of the PLC software should show evidence of achieved software testing.

2.1 Programmable Logic Controllers

A programmable logic controller is type of microprocessor-based controller [5]. It manages processes and controls machines using a programmable memory which is designed to be operated by engineers with limited knowledge programming languages. The modularization of the PLC allows changes in the control system just by using new instructions without any hardware changes, which makes PLCs cost efficient and flexible [5]. This makes PLCs highly customizable and allows mixed types of I/O devices according to applications requirements and is often used in different industrial environments [6]. However, due to its flexibility, PLCs lack standardization and cannot fulfill the standard set by the International Electrotechnical Commission (IEC) within the IEC 61131-3 [3] standard entirely. Each PLC usually provides a document describing which parts of the IEC 61131-3 standard are implemented and which parts of the IEC 61131-3 standard are not covered [6].

A PLC usually consists of the following constituent parts (mirrored in Figure 2.1) [5]:

- A central processing unit (CPU)
- A power supply
- A programming device
- A memory unit
- A input and output sections
- A communications interface

The CPU which contains the microprocessor interprets the input signals and carries out control actions accordingly. The program and signals which the CPU has to interpret are developed in the programming device where users can enter the necessary program into the memory unit. The memory unit contains the control actions which the CPU can carry out when needed. It also stores input and output information. The input interface receives information from external devices (e.g. temperature sensors or flow sensors) and communicates through either discrete, digital or analog signals. The output interface also communicates through discrete, digital or analog signals but
instead of receiving information from external devices it delivers signals to external devices. To communicate with other PLCs [5], the communication interface is used to receive and transmit data.

A program running on a PLC usually executes in a cyclic loop. Each loop has three phases which begins with a reading phase where all inputs are read and stored accordingly. This phase is followed by an execution phase where all computations are performed and are then passed to the last writing phase which updates the outputs. The processor primarily interprets logic and switching operations to control input devices. The program that the CPU executes must be written in one of the five languages defined by the International Electrotechnical Commission (IEC) within the IEC 61131-3 standard [3]. An FBD program for a PLC could look as shown in Figure 2.1. This program is used to handle the train functionality of calculating the sum of the available break effort and achieved brake effort. It also determines if output shall be set and distributed to all parts of the train.

In the program shown in Figure 2.1, the DIV block divides input 2 with constant 1 and sends the result forward to the next block. GT checks if the user input is greater than the constant input and it outputs a signal of value 1 if the input is greater and 0 if it is less than the constant. The AND block outputs a signal of value 1 if both inputs are equal to 1 and outputs 0 if either of the inputs are 0. An example of the generated outputs using certain test cases are shown in Table 2.1.
### 2.2 Combinatorial Testing

Numerous techniques have been developed to help software testers in automatically generating tests [7]. In the last two decades different combinatorial testing techniques were proposed in order to automate the test generation process. Combination strategies generate test cases that covers all the interactions of variables at some level of interaction by allowing the selection of values and their combination according to a certain algorithm to produce a complete test case [1]. Some of these combination strategies include: All Combination (AC), Each Choice (EC), Orthogonal Array (OA), Automatic Efficient Test Generator (AETG) and Base Choice (BC). The result of using a combination strategy is a test case, a combination of variable values that are chosen depending on variable ranges and the strategy used. Each test case is executed one at a time on the program under test. A collection of test cases defines a test suite.

### 2.3 Base Choice Criteria

Base Choice (BC) [8] is a combinatorial testing technique used to automatically generate test cases. The base test case may be determined by a certain criterion, like the simplest value. A criterion suggested by Ammann and Offutt [8] is the most likely value from the point of view of the user. This value could be determined by the tester. By using BC, one is able to generate test cases by choosing a range for each input variable and a base (default) choice value. The test cases are then generated by using the base choice value for all variables except one, which goes through all non-base choice values and then it continues with the next variable. The selected base choice is typically either an expected and common input value or a default value.

<table>
<thead>
<tr>
<th>Test case #</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
<th>Input 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.2: Example of generated test using BC

For example, tests generated using BC strategy for the program shown in Figure 2.2 are obtained...
by using the ranges and base choices for each input, as follows:

- Range $[1, 5]$ for (input 1, 2 and 3)
- Base choice values for all inputs. (1 for input 1, 2 for input 2 and 3 for input 3)
- Input 4 has the range $[0,1]$
- Base choice value 0 for input 4

The test cases are generated by using all non-base choice values for Input 1 which had base choice 1. In combination with these values the other inputs uses their set base choice. The generation then continues by repeating the procedure for all other inputs. The generated test suite containing a set of 17 test cases is shown in Table 2.2.
Chapter 3

Methodology

We designed a case study for evaluating the use of base choice criteria for testing PLC software. In this chapter we show how we selected the programs and performed a case study.

3.1 Problem Formulation

Software testing is important for ensuring the quality of a program. Therefore, it is of utmost interest to investigate how base choice testing can be used to improve testing industrial PLC software in terms of efficiency and effectiveness. Bombardier Transportation is a company producing PLC software and the process of testing PLC software is mainly based on manually testing the software based on both the program specification and the program code.

Since manual testing can be time consuming, it may be of value to consider automated test generation approaches (i.e., base choice). Researchers have shown the practical application of different combinatorial test generation techniques (e.g., base choice (BC) strategy). In particular, recent empirical studies [1, 2] showed the value of using base choice coverage and its fault detection capability. However, there is no conclusive evidence on how base choice could be used for industrial software. This is particularly true for PLC software, where manual testing is rigorous and effective. Three questions arose from this:

- RQ1: How does BC and manual tests compare in terms of achieved code coverage?
- RQ2: Are BC tests less costly to use than manual tests?
- RQ3: Are BC tests more effective in fault detection than manual tests?

We considered an industrial project provided by Bombardier Transportation AB containing PLC programs and for each selected program, we executed the tests produced by both manual testing and base choice testing and collected the following measures: code coverage in terms of achieved decision coverage, the number of tests and the mutation score as a proxy for fault detection. In order to calculate the mutation score, each test was executed on the mutated version of the original TCMS program to determine whether it detects the injected fault or not. The following sections describe in detail the case study process.

3.2 Case Study Process

A research methodology is a structured and systematic process that answers a research question. The research started with the opportunity of using BC testing in PLC software development, and ended with performing an empirical evaluation [9] and analysis of the results. The case study process, mirrored in Figure 3.1, started with the formulation of the research problem that was broken down into the three research questions (i.e., RQ1, RQ2, and RQ3) which are investigated in this thesis. The study began with the problem of using BC for PLC software and continued with further research about this problem in terms of current state of the art and state of the
practice. We gathered knowledge on how to use base choice criteria and how to use it in practice through existing scientific studies and based on the results found we derived hypotheses that were formulated as research questions. The questions were answered using a case study.

![Figure 3.1: Research methodology.](image)

An evaluation with regard to BC’s fault detection capability and its success in meeting code coverage was performed using a case study. In addition, in the same case study, we compared BC with Random tests of the same size as BC which was used as a baseline and helped in the evaluation of the results. BC was compared to manual testing in terms of number of tests and fault detection using mutation analysis, as mirrored in Figure 3.2. The resulted data was analyzed using descriptive statistics. Descriptive statistics were used to visualize and provide an overview of the data, e.g., in plots and graphs. We used descriptive statistics to briefly compare and evaluate data to find patterns, e.g., to compare if two data sets are very different from each other.

The set of programs used and tested in this thesis are written in IEC 61131-3 FBD language [3]. A combinatorial testing tool [10] was used to generate test cases that was developed previously, which is able to generate test cases for IEC 61131-3 programs using base choice and Random testing. The resulting tests were used to run the programs and measure fault detection, code coverage in terms of branch coverage and the cost (in terms of generation time and number of test cases). The measurements were obtained using a simulation and execution framework provided by Bombardier Transportation. The tests created manually by engineers at Bombardier Transportation were also measured using the same execution framework. The results were presented in an explanatory and quantitative manner [9]. The research methodology, the case study process, the results of the experiment and the conclusion and discussion are reported in the following sections.

### 3.2.1 Selection of Programs

Bombardier Transportation AB provided us programs from the Train Control Management System (TCMS). TCMS is a distributed embedded system with multiple types of software and hardware subsystems, and takes care of much of the operation and safety-related functionality of the train. TCMS runs on Programmable Logic Controllers (PLC) which are real-time systems used in numerous industrial domains, i.e., nuclear plants and oil industry. The system allows for integration of control and communication functions for high speed trains and contains all functions controlling the train. A total number of 86 programs were provided by Bombardier Transportation. We used a simple criterion to select the subject programs for the case study, which involved identifying
3.2.2 Combinatorial Testing Tool

The combinatorial testing tool (CTT) used in this thesis was chosen because it works with PLC software and it supports both base choice criteria as well as Random testing. Figure 3.3 illustrates the functionality of CTT. However, some of the programs used in the case study included variables that were not originally supported in CTT. The software is available online and can be found on github [10]. As the first contribution of this thesis we implemented the support for new type variables. The variables in the PLC programs had to be translated into C# since the CTT was written in C# programming language. We based our translation on the variable type transformation developed by Kunze [6]. Both the previously supported data types and the newly supported data types are shown in Table 3.1.

<table>
<thead>
<tr>
<th>Originally supported variable types in CTT</th>
<th>Newly implemented variable types</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>INT</td>
</tr>
<tr>
<td>BOOL</td>
<td>BOOL</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>-</td>
<td>UINT</td>
</tr>
<tr>
<td>-</td>
<td>USINT</td>
</tr>
<tr>
<td>-</td>
<td>UDINT</td>
</tr>
</tbody>
</table>

Table 3.1: Supported variable types in CTT.

3.2.3 Test Generation

Manual test suites were collected by using the test data available. In testing PLC programs in TCMS, the engineering processes of software development are performed according to safety standards and regulations. For each program we collected a manual test suite created by industrial engineers in Bombardier Transportation AB. Executing each test suite on TCMS is supported by an
The test suites collected in this thesis were based on functional specifications expressed in a natural language.

In addition, we automatically generated test suites using BC. BC testing was performed by the author of this study by using predetermined ranges and base values for each input variable. The ranges were obtained by looking directly in the comments contained in each program code. A criterion suggested by Ammann and Offutt in the original definition of base choice is the most likely value from the point of view of the tester. In order to collect realistic data, we asked one test engineer from Bombardier Transportation, responsible for testing PLC software similar to the one used in this study, to identify the default value of each input variable based on the predetermined ranges. The test engineer provided base choice values for all input variables. Interestingly enough, one default value (i.e. 1.0 for variables of type float, 1 for variables of type integer and true for boolean variables) was provided for all programs. The engineer argued that programs in TCMS contain default inputs that are usually used for testing the basic functionality of the program.

In addition, we generated tests using a Random testing strategy. We wanted to use Random testing as a baseline for comparison. In order to be able to determine how many test cases to use for the Random testing strategy, we measured the time taken to generate a certain amount of test cases using both BC and Random testing and compare the generation time. With this measurement we wanted to observe if there is any significant difference in terms of generation time between BC and Random. We argue that if there is a significant difference, Random tests should not be generated using the number of tests of BC as the stopping criteria but instead by using the generation time of BC. We implemented in CTT a visualization module automatically showing the number of test cases generated. This allowed an easier configuration of the number of test cases that should be generated using Random testing. The generation time measurements between BC and Random for one PLC program can be seen in Table 3.2.

There is no significant difference (see Table 3.2) in the generation time between test generated using BC and Random. We decided, based on these results, that Random tests should be generated based on the number of test cases generated using BC. In other words, we generated tests with the same number of test cases as BC, as this is a realistic scenario for a baseline approach.

### 3.2.4 Metrics

In this study, several metrics have been measured in order to answer our research questions: code coverage is operationalized using the decision coverage criterion, mutation score for estimating the fault detection capability, and the number of test cases as a cost estimate.
Decision Coverage

Code coverage criteria are used in software testing to evaluate how good test cases are [11]. These criteria are usually used to verify the extent to which the code has been evaluated by each test case. In this thesis, code coverage is measured using the decision coverage criterion. For the programs selected in this study and developed by Bombardier Transportation AB, engineers developing PLC software indicated that their testing process involves achieving high decision coverage. In the context of traditional programming languages (e.g., C#), decision coverage is usually named branch coverage. A decision coverage score was obtained for each test suite. A test suite satisfies decision coverage if running the test cases causes each branch in the PLC program to have the value `true` at least once and the value `false` at least once. Decision coverage was previously used by Enoiu et al. [12] to improve testing of PLC software. In this thesis we used the execution framework to collect the decision coverage achieved by each test suite.

Fault Detection

Mutation analysis is used to measure fault detection of different testing strategies [13]. Instead of using real faults, mutation analysis injects faults into the software creating faulty versions [11]. These faults are systematically seeded throughout the program for analyzing the fault detection capability of a test suite [14]. Mutation analysis has been shown to be useful for experimentation and a good proxy for fault detection since it can be used in cases where real faults are not available. The number of mutants created for each program used in the experiment can be shown in Table 3.3.

---

In software testing literature, a software fault is a static defect in the software, while a software error is an incorrect internal state that is the manifestation of some fault. A software failure refers to the incorrect behavior with respect to the requirements or other description of the expected behavior.

---

Table 3.2: Generation time in milliseconds of BC and Random using CTT for one PLC program.

<table>
<thead>
<tr>
<th># Test</th>
<th>BC</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3153</td>
<td>3153</td>
</tr>
<tr>
<td>2</td>
<td>3166</td>
<td>3333</td>
</tr>
<tr>
<td>3</td>
<td>3187</td>
<td>3228</td>
</tr>
<tr>
<td>4</td>
<td>3260</td>
<td>3219</td>
</tr>
<tr>
<td>5</td>
<td>3191</td>
<td>3151</td>
</tr>
<tr>
<td>6</td>
<td>3252</td>
<td>3219</td>
</tr>
<tr>
<td>7</td>
<td>3169</td>
<td>3199</td>
</tr>
<tr>
<td>8</td>
<td>3302</td>
<td>3277</td>
</tr>
<tr>
<td>9</td>
<td>3114</td>
<td>3148</td>
</tr>
<tr>
<td>10</td>
<td>3198</td>
<td>3292</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>3199.2</td>
<td>3221.9</td>
</tr>
</tbody>
</table>

---
### Table 3.3: Number of mutants created for each program used in the experiment.

<table>
<thead>
<tr>
<th>Program</th>
<th># Mutants</th>
<th>Program</th>
<th># Mutants</th>
<th>Program</th>
<th># Mutants</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>38</td>
<td>P16</td>
<td>38</td>
<td>P31</td>
<td>8</td>
</tr>
<tr>
<td>P2</td>
<td>11</td>
<td>P17</td>
<td>33</td>
<td>P32</td>
<td>19</td>
</tr>
<tr>
<td>P3</td>
<td>6</td>
<td>P18</td>
<td>19</td>
<td>P33</td>
<td>94</td>
</tr>
<tr>
<td>P4</td>
<td>81</td>
<td>P19</td>
<td>63</td>
<td>P34</td>
<td>31</td>
</tr>
<tr>
<td>P5</td>
<td>22</td>
<td>P20</td>
<td>19</td>
<td>P35</td>
<td>29</td>
</tr>
<tr>
<td>P6</td>
<td>68</td>
<td>P21</td>
<td>30</td>
<td>P36</td>
<td>45</td>
</tr>
<tr>
<td>P7</td>
<td>38</td>
<td>P22</td>
<td>9</td>
<td>P37</td>
<td>10</td>
</tr>
<tr>
<td>P8</td>
<td>11</td>
<td>P23</td>
<td>15</td>
<td>P38</td>
<td>52</td>
</tr>
<tr>
<td>P9</td>
<td>39</td>
<td>P24</td>
<td>7</td>
<td>P39</td>
<td>21</td>
</tr>
<tr>
<td>P10</td>
<td>34</td>
<td>P25</td>
<td>27</td>
<td>P40</td>
<td>25</td>
</tr>
<tr>
<td>P11</td>
<td>39</td>
<td>P26</td>
<td>38</td>
<td>P41</td>
<td>18</td>
</tr>
<tr>
<td>P12</td>
<td>3</td>
<td>P27</td>
<td>100</td>
<td>P42</td>
<td>20</td>
</tr>
<tr>
<td>P13</td>
<td>12</td>
<td>P28</td>
<td>76</td>
<td>P43</td>
<td>29</td>
</tr>
<tr>
<td>P14</td>
<td>33</td>
<td>P29</td>
<td>93</td>
<td>P44</td>
<td>21</td>
</tr>
<tr>
<td>P15</td>
<td>47</td>
<td>P30</td>
<td>58</td>
<td>P45</td>
<td>31</td>
</tr>
</tbody>
</table>

**Number of Test Cases**

There are several ways of measuring cost efficiency. Cost measurement of software testing contains direct and indirect type of cost, measuring directly the test creation, the test execution and the checking of test results. As the programs used in this thesis were already developed and we did not have any information about the cost of performing manual testing we use a surrogate measure of the cost, the number of test cases. This information was obtained automatically by using the execution framework provided by Bombardier Transportation. In this thesis we consider that the cost is related to the number of test cases. The higher the number of tests cases, the higher are the respective costs.

#### 3.2.5 Descriptive Statistics

The resulted data itself was statistically analyzed using descriptive statistics. Descriptive statistics was used to visualize and provide an overview of the data, e.g., in plots and graphs. For each measure we list the detailed statistics on the obtained results, like minimum values, median, mean and standard deviation.

For each program under test and each testing technique (i.e., base choice (BC), manual testing (Manual) and Random), we collected a test suite. These sets of tests were used to conduct the experimental analysis. For each test suite produced, we derived three distinct metrics: mutation score, decision coverage and number of test cases. These metrics form the basis for our analysis towards the goal of answering the research questions. In our analysis we used the R statistical tool.

#### 3.3 Execution Framework

To perform the experiment, a way of measuring decision coverage, mutation score and cost was needed. This was done using an execution framework from Bombardier AB. The tool included FBDMutator.jar to generate mutations and PLC.jar to measure the needed data. The FBDMutator used an XML file to generate mutations of the program which would then be used by the PLC.jar during the execution. By injecting faults into a program it is possible to know how many of these faults were found and not found, which allows a measuring of fault detection.
Chapter 4

Results and Analysis

In this section the data collected from the experiment is presented and analyzed. In Table 4.1 we present the mutation scores, coverage results and the number of test cases in each collected test suite (i.e., manually created test suites by industrial engineers, base choice test suites, and Random test suites of the same size as the ones created using BC). This table lists the minimum, median, mean, maximum and standard deviation values. The data provided are the results from the case study performed on 45 PLC programs. Overall, Random and BC performed similarly in all three measured metrics. As an example, for both decision coverage and mutation score, the dispersion of BC and Random is rather high which can be shown by the standard deviation results. This shows that the use of BC and Random can be unreliable and would question the use of these strategies. In average BC achieved decision coverage of 69.53% in comparison with 71.74% for Random, which shows no substantial improvement in the fault finding capability obtained by BC test suites over their Random counterparts of the same size.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Test</th>
<th>Min</th>
<th>Median</th>
<th>Mean</th>
<th>Max</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutation Score (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>44.44</td>
<td>95.00</td>
<td>88.90</td>
<td>100.00</td>
<td>14.38</td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>9.09</td>
<td>80.77</td>
<td>69.53</td>
<td>100.00</td>
<td>33.72</td>
<td></td>
</tr>
<tr>
<td>Rand</td>
<td>6.06</td>
<td>86.84</td>
<td>71.74</td>
<td>100.00</td>
<td>31.47</td>
<td></td>
</tr>
<tr>
<td>Decision Coverage (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>63.64</td>
<td>100.00</td>
<td>97.38</td>
<td>100.00</td>
<td>6.80</td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>50.00</td>
<td>85.71</td>
<td>83.10</td>
<td>100.00</td>
<td>15.35</td>
<td></td>
</tr>
<tr>
<td>Rand</td>
<td>50.00</td>
<td>90.00</td>
<td>86.23</td>
<td>100.00</td>
<td>14.54</td>
<td></td>
</tr>
<tr>
<td># Tests</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>2.00</td>
<td>7.00</td>
<td>12.84</td>
<td>56.00</td>
<td>12.40</td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>4.00</td>
<td>11.00</td>
<td>15.13</td>
<td>76.00</td>
<td>12.83</td>
<td></td>
</tr>
<tr>
<td>Rand</td>
<td>4.00</td>
<td>11.00</td>
<td>15.13</td>
<td>76.00</td>
<td>12.83</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Experiment Results.

4.1 Decision Coverage

As seen in Figure 4.1, manual tests performed achieved high decision coverage with the majority of tests obtaining 100% decision coverage. Test generation based on BC achieved lower fault detection than manual testing and a wider spread of data, this being visible from the standard deviation values in Table 4.1. The data for manual testing had a standard deviation of 6.80% compared to BC with 15.35%. The Random testing strategy showed a higher average code coverage score than BC but also had a wider spread of values from 50% to 90%. Out of the three testing strategies, BC had the lowest median code coverage score, 85.71% compared to 100.00% code coverage score achieved by manual tests and 90.00% decision coverage score for Random testing. For the programs considered in this thesis, manual testing achieves better decision coverage and fault detection compared to BC testing performance in terms of decision coverage for industrial programs written in IEC 61131-3 language. Each test case in BC test suites are derived from a certain base choice value, meaning that certain numbers appear more frequent than others. This
could explain why BC did not perform as well as manual tests, since certain paths taken in the program could be less often visited or be excluded all together.

Figure 4.1: Decision coverage achieved by each testing approach. Manual stands for manual testing, BC stands for Base Choice and Random stands for Random tests of the same size as BC tests; boxes span from 1st to 3rd quartile, black middle lines mark the median and the whiskers extend up to 1.5x the inter-quartile range and the circle symbols represent outliers.

While manual tests performed better than BC on average in terms of achieved decision coverage, we were interested in looking closely (see Figure 4.2) at the number of tests where manual testing was better, equal or worse than BC. Interestingly enough, 72% of all manual tests achieved better decision coverage than BC tests (9% of all BC tests achieving higher decision coverage than manual tests). This shows that manual tests are significantly better than BC tests. This could be due to the selection of the base value for BC testing. In the next chapter we are investigating this further.

Answer RQ1: Manual test suites perform better than test suites generated using base choice strategy in terms of decision coverage.
4.2 Fault Detection

The quality of a test suite in terms of fault detection was measured using mutation analysis with faults automatically injected into each of the 45 programs. Manual tests achieved a higher average mutation score (i.e., 88.90% for manual testing) than either BC tests (i.e., 69.53%) and Random tests (i.e., 71.74%). As shown in Table 4.1, the data for BC and Random testing had similar standard deviations but quite different from manual testing data. The results of the fault detection comparison are also illustrated in Figure 4.3 as box-plots. Manual testing is with 20% in average better in terms of fault detection than BC testing. This can be explained by the fact that manual tests are written by engineers with experience in testing PLC software, while base choice tests are only based on a certain combination of inputs that is not exercising certain parts of the program behavior. For example, BC is not taking into account programs that contain timers and state-full blocks in which the inputs need to be kept for a certain amount of time before the output can change.

Answer RQ3: **Manual test suites are considerably more effective in fault detection than test suites using base choice strategy.**

Even though manual testing performed better in average than BC testing, from the total number of tests only 53% showed better mutation score (see Figure 4.4). Test suites generated based on BC performed equally or better than manual testing for 47% of tests executed (for 16% of the tests, BC is better in terms of fault detection while for 31% of the tests BC and manual testing achieve the same mutation score). The results can be explained by the fact that manual test suites are written by experienced engineers for an already developed train.
Figure 4.3: Mutation score achieved by each testing approach. Manual stands for manual testing, BC stands for Base Choice and Random stands for Random tests of the same size as BC tests; boxes spans from 1st to 3rd quartile, black middle lines mark the median and the whiskers extend up to 1.5x the inter-quartile range and the circle symbols represent outliers.

Figure 4.4: Mutation Score comparison between Manual and Base Choice Testing showing which technique performed better worse or equal in terms of mutation score.

4.3 Number of Test Cases

In this section we show the comparison in terms of number of test cases between manually created tests and tests created using a combinatorial testing tool [10]. Obviously, BC and Random testing result in the same number of test cases created for each program (see Figure 4.5). The difference in number of tests between manual and BC is not significantly large. Manually created tests have in average 12.84 test cases while BC generated contain in average 15.13 test cases. Based on this measure alone we can say that manual tests are similar in terms of the number of tests (2.29 test cases less in average for BC testing).
Answer RQ2: The use of base choice testing results in similar number of tests as manual testing.

![Box plot showing number of test cases for each testing approach.](image)

Figure 4.5: Number of test cases created for each testing approach. Manual stands for manual testing, BC stands for Base Choice and Random stands for Random tests of the same size as BC tests; boxes spans from 1st to 3rd quartile, black middle lines mark the median and the whiskers extend up to 1.5x the inter-quartile range and the circle symbols represent outliers.

From the results shown in Table 4.1 and Figure 4.6 we can see that there is no significant difference in the number of test cases created. Manual testing used fewer test cases in 53% of the programs. In 42% of the cases BC tests were shorter than manual tests and in 5% of the cases manual and BC tests had the same number of tests. This shows that there is no considerable difference in cost efficiency in terms of number of tests between BC and manual tests even if there is a significant difference in terms of achieved mutation score and decision coverage.

![Pie chart showing test cases comparison.](image)

Figure 4.6: Number of test cases comparison between Manual and Base Choice Testing showing which technique performed better worse or equal in terms of number of tests generated.
Chapter 5
Discussion and Validity Threats

This section provides a brief discussion on the results of this thesis. The results are discussed in order to try and explain some of the results obtained. In addition, the limitations and the validity threats are showed in this chapter.

5.1 Discussion

For each of the research questions investigated in the case study the results showed that manually created tests performed better for all metrics considered. Manual testing achieves better decision coverage and fault detection than both BC and Random testing strategies. The results in cost efficiency in terms of number of test cases, for all three strategies, showed no significant difference. The results also showed that BC testing achieved for some of the programs lower decision coverage and fault detection than Random tests of the same size as BC.

Manual test suites were created by engineers at Bombardier Transportation AB with experience in testing of PLC programs. Our results support the hypothesis manual test suites are of high quality. One explanation for the difference in fault detection between manual and BC testing is that when using BC testing strategy, ranges and base choice values for each input variable had to be set prior to the test generation. In reality these ranges and base choice values should be set by the engineers programming and testing PLC programs at Bombardier Transportation AB. As this was practically unfeasible to perform during this thesis, we obtained data for BC testing from a test engineer that has knowledge of PLC programming but not to the extent of an engineer implementing a PLC program. The test engineer provided information on the base values for the whole TCMS project and therefore the data provided may not have been optimal for the creation of high quality BC test suites. To show that there is some evidence to this fact, we created tests, mirrored in Table 5.1 and Table 5.2, using BC for one program using the base choice provided by the test engineer and by the author of this thesis after carefully checking the behavior of the program under test.

For the program used in this example, the base choice value for the original test suite is 1 for all input parameters. We decided to create another test suite using BC and used 0 as a base choice for all inputs. According to the results shown in Table 5.3, just by changing the base choice value from 1 to 0 the achieved decision coverage increased from 85.71 % to 100.00 % and the mutation score increased from 15.79% to 92.11%. This is a significant difference that can obviously explain why BC is much worse in terms of fault detection than manual tests and even Random tests of the same size. Therefore, we argue that BC could potentially perform better if the base choice value can be selected much more carefully for each program. Even so, the base choice values provided by the test engineer are not selected by a person with no experience in PLC testing and the results are showing that BC used in the same manner it was used in this thesis is not particularly good at fault detection when compared to either manual testing or Random testing.

Even if BC did not perform as well as the other testing strategies used in the experiment, there are several studies that show the significance of BC [2]. In Grindal et al.’s study, BC detected 99 % of all detectable faults compared to 69.53 % achieved in the experiment performed for this thesis. This study also suggests that BC is preferable to use for programs that are user-oriented.
Table 5.1: Test Suite for BC using the base choice provided by the test engineer.

<table>
<thead>
<tr>
<th>Input variable</th>
<th>Range(s)</th>
<th>Base Choice value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT 1</td>
<td>-500; -56 to -54; 0; 120 to 128; 500</td>
<td>1</td>
</tr>
<tr>
<td>BOOL 1</td>
<td>0 to 1</td>
<td>1</td>
</tr>
<tr>
<td>BOOL 2</td>
<td>0 to 1</td>
<td>1</td>
</tr>
<tr>
<td>BOOL 3</td>
<td>0 to 1</td>
<td>1</td>
</tr>
<tr>
<td>BOOL 4</td>
<td>0 to 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.2: Test Suite for BC using the base choice selected by the author of this thesis.

<table>
<thead>
<tr>
<th>Input variable</th>
<th>Range(s)</th>
<th>Base Choice value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT 1</td>
<td>-500; -56 to -54; 0; 120 to 128; 500</td>
<td>0</td>
</tr>
<tr>
<td>BOOL 1</td>
<td>0 to 1</td>
<td>0</td>
</tr>
<tr>
<td>BOOL 2</td>
<td>0 to 1</td>
<td>0</td>
</tr>
<tr>
<td>BOOL 3</td>
<td>0 to 1</td>
<td>0</td>
</tr>
<tr>
<td>BOOL 4</td>
<td>0 to 1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.3: Comparison between two BC test suites each with different base values.

<table>
<thead>
<tr>
<th>Test</th>
<th>Decision Coverage</th>
<th>Mutation Score</th>
<th># Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using BC value 1</td>
<td>85.71</td>
<td>15.79</td>
<td>18.00</td>
</tr>
<tr>
<td>Using BC value 0</td>
<td>100.00</td>
<td>92.11</td>
<td>18.00</td>
</tr>
</tbody>
</table>

Obviously, the programs used in the experiment for this thesis were not user-oriented since the programs are used in a train control management system.

5.2 Threats to Validity

The experiment was performed using industrial programs written in FBD which is one of five languages according to the IEC 61131-3 standard [3] and the results cannot be generalized for all PLC software. We argue that the results are representative of programs created in an industrial environment in an embedded system controlling a train. More studies are needed to generalize the results.

The generated tests using BC and Random testing were created using the combinatorial testing tool (CTT) previously developed for PLC software [10]. There are many other tools for combinatorial testing which may give different results. We argue that CTT uses already known and available algorithms that should perform similarly to other tools like ACTS [15].

We used mutation analysis as a proxy for fault detection effectiveness of BC, Random and manual test strategy. We did not use real faults to evaluate these strategies and this fact may affect the outcome of the results. However, mutants have been shown to be a valid substitute for real faults [14], which would support the use of mutation analysis for our purposes.
Chapter 6

Related Work

Ammann and Offutt [8] defined a combinatorial coverage criterion for category-partition test specification in 1994. They identified a process to produce tests that satisfied this criterion. The purpose of this criterion was to relieve programmers from some of the burden of testing the software such that they can concentrate on other tasks. This paper [8] is the first to introduce the Base Choice testing strategy and suggests this method for automatically testing software. In the same paper as BC was introduced [8], Ammann and Offutt also introduced another testing strategy called Each Choice (EC). The testing strategy requires each value of each parameter to be included in at least one test case.

One testing strategy known as Orthogonal Array (OA) was first introduced to be used in software testing by Mandl in 1985 [16]. Mandl suggested OA as an alternative to exhaustive testing, i.e. a testing method using all possible data combinations, and/or Random testing. The problem with exhaustive testing is that the test may be extremely large. Random tests are often a bit more manageable in size but the problem lies in its reliability, which is one of the most important aspects in software testing in terms of ensuring the quality of the program.

In 1994 a testing tool was introduced called Automatic Efficient Test Generator (AETG) [17] which uses algorithms inspired by ideas from statistical experimental design theory. It is used to minimize the number of test needed for a specific level of test coverage of the input test space. This tool is proven to be superior when compared to exhaustive testing. In Cohen et al.s paper, AETG is compared to experimental design techniques and in particular Orthogonal Array (OA). Studies show that AETG have several advantages over OA, making AETG a more reliable method of testing.

In Grindal et al.s [2] research study, the results show that BC achieves a high percentage of fault detection while using fewer tests than other combinatorial strategies such as OA and AETG. However, BC did require more tests than EC and showed lower fault detection scores than EC strategy. Although BC may not be as efficient as EC in terms of number of faults detected per test, BC may yet be more valuable because of its ability to detect faults that EC may not detect. Grindal et al. [2] also states that some of the faults detected by EC was by coincidence and the use of EC may be too unpredictable for real usage. For the programs used in this study, EC detected 90% of faults, while BC managed to achieve 99% fault detection with only one missed fault. OA and HPW managed to detect the fault that BC missed, however, these two strategies had 98% fault detection.

There are different combinatorial testing tools which support BC test generation. One of these tools is called Advanced Combinatorial Testing System (ACTS) [15], which uses t-way combinatorial test generation with several features. Another tool is one developed by several students at Mälardalen University [10]. This tool supports both BC and Random test generation. This specific tool was used during the test generation for the experiments performed in this thesis.

To our knowledge, no previous studies has been made comparing the base choice test strategy to manual testing. There is a need to further study how good BC is at detecting faults compare to manual testing performed by industrial engineers.
Chapter 7

Conclusion

In this thesis we evaluate base choice criteria and compare it with manual and Random testing strategies for industrial PLC programs [3]. A case study was performed based on 45 programs from a recently developed safety-critical embedded software and manual test suites produced by industrial professionals. Using this case study, we tried to answer three research questions. The purpose of the first research question (RQ1: How does BC and manual tests compare in terms of achieved code coverage?) was to determine the quality in terms of achieved decision coverage. Our results show that manual tests are considerably better than test suites created using BC testing strategy.

To further investigate the quality of BC testing, the second research question was posed (RQ2: Are BC tests less costly to use than manual tests?). The cost was measured in terms of number of test cases used for each test suite. This thesis shows that manual tests were similar in terms of number of test cases to base choice tests.

The third and final research question (RQ3: Are BC tests more effective in fault detection than manual tests?) involves the measurement of fault detection using mutation analysis. Manual testing achieved better fault detection than BC.
Chapter 8

Future Work

According to the results of this thesis, manual testing is preferable to BC since it is much better in terms of achieved decision coverage and fault detection. There could be several reasons for this: it could be of value to investigate this further by performing the same experiment with new ranges and especially new base choice values that are chosen by an expert for each single program. Hypothetically, this could improve the results considerably for both decision coverage and fault detection.

The test cases generated for each test suite are based on a single base choice value for each input. This could be improved by introducing an option to use several base choice values which can be used for certain programs.

The results for cost efficiency were only taking into account the number of test cases for each test suite which is only a small part of the whole testing cost. To represent the total cost for the use of a specific testing strategy we need to measure the time the engineer spends using BC or manually testing each program. Therefore, it could be of interest to measure these factors in a similar study.
Bibliography


