MODEL-BASED INTEGRATION TESTING TECHNIQUE USING FORMAL
FINITE STATE BEHAVIORAL MODELS FOR COMPONENT-BASED
SOFTWARE

ABUBAKAR ELSAFI ALI AHMED

A thesis submitted in fulfilment of the requirements for the award of the degree of Doctor of Philosophy (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

AUGUST 2017
To my beloved parents, brothers and sisters who always support and encourage me in the good times as well as the bad times
ACKNOWLEDGEMENT

First and foremost, my praises and thanks to Almighty Allah (S.W.T), the Most Gracious the Most Merciful, who gave me the health, strength, knowledge, encouragement and patience to accomplish this research. May the peace and blessings of Allah be upon our Prophet Mohammed (S.A.W).

I wish to express my deep gratitude to my supervisor Assoc. Prof. Dr. Dayang Norhayati binti Abang Jawawi, Deputy Dean (Academic), Faculty of Computing, Universiti Teknologi Malaysia (UTM) for her guidance to the right way and continuous support which helped me to stay focused from the beginning to the end of the research process. I would like to thank her because she has never been too busy to keep an eye on my progress in spite of her numerous obligations. She has greatly helped me in a lot of ways throughout this study. Again, I owe her my deepest thanks.

In addition, I am grateful to my colleagues at Embedded & Real-Time Software Engineering Laboratory (EReTSEL) for the useful discussions, guidance, advice and knowledge sharing in this field. Without their contribution, interest and guidance I would not be able to complete this thesis.

I would like to mention my beloved parents, brothers and sisters who has always believed in me and had always supported me during my stay in Malaysia and eagerly await my arrival back home. Their prayers have always been a source of inspiration and encouragement for me. I thank you all from the core of my heart because your prayers and well wishes made it possible for me.

There is a common Arabic saying: "I am indebted forever to whoever teaches me something, even if it is a single letter", so my thanks are due to everyone who taught me something during my PhD study. Finally, my appreciation also goes to all my friends here at UTM.
ABSTRACT

Many issues and challenges could be identified when considering integration testing of Component-Based Software Systems (CBSS). Consequently, several research have appeared in the literature, aimed at facilitating the integration testing of CBSS. Unfortunately, they suffer from a number of drawbacks and limitations such as difficulty of understanding and describing the behavior of integrated components, lack of effective formalism for test information, difficulty of analyzing and validating the integrated components, and exposing the components implementation by providing semi-formal models. Hence, these problems have made it ineffective to test today's modern complex CBSS. To address these problems, a model-based approach such as Model-Based Testing (MBT) tends to be a suitable mechanism and could be a potential solution to be applied in the context of integration testing of CBSS. Accordingly, this thesis presents a model-based integration testing technique for CBSS. Firstly, a method to extract the formal finite state behavioral models of integrated software components using Mealy machine models was developed. The extracted formal models were used to detect faulty interactions (integration bugs) or compositional problems between integrated components in the system. Based on the experimental results, the proposed method had significant impact in reducing the number of output queries required to extract the formal models of integrated software components and its performance was 50% better compared to the existing methods. Secondly, based on the extracted formal models, an effective model-based integration testing technique (MITT) for CBSS was developed. Finally, the effectiveness of the MITT was demonstrated by employing it in the air gourmet and elevator case studies, using three evaluation parameters. The experimental results showed that the MITT was effective and outperformed Shahbaz technique on the air gourmet and elevator case studies. In terms of learned components for air gourmet and elevator case studies respectively, the MITT results were better by 98.14% and 100%, output queries based on performance were 42.13% and 25.01%, and error detection capabilities were 70.62% and 75% for each of the case study.
ABSTRAK

Pelbagai isu dan cabaran dapat dikenal pasti apabila mempertimbangkan ujian integrasi bagi Perisian Sistem Berasaskan Komponen (CBSS). Oleh yang demikian, beberapa penyelidikan telah dilaksanakan dalam kajian lepas yang bertujuan untuk memudahkan ujian integrasi bagi CBSS. Namun, kajian tersebut mengalami beberapa kelemahan dan batasan seperti kesukaran dalam memahami dan menggambarkan tingkah laku komponen bersepadu, kekurangan formalisme yang berkesan bagi ujian maklumat, kesukaran dalam menganalisis dan mengesahkan komponen bersepadu, dan pendedahan pelaksanaan komponen dengan menyediakan model separa formal. Oleh itu, masalah ini telah membuatnya tidak berkesan untuk menguji kompleks CBSS moden pada hari ini. Bagi menangani masalah tersebut, pendekatan berasaskan model seperti Ujian Berasaskan Model (MBT) cenderung menjadi mekanisme yang sesuai dan boleh menjadi penyelesaian yang berpotensi untuk digunakan dalam konteks ujian integrasi CBSS. Sehubungan itu, kajian ini membentangkan teknik ujian integrasi berasaskan model untuk CBSS. Pertama, satu kaedah untuk mengekstrak model formal tingkah laku keadaan terhingga bagi integrasi komponen perisian bersepadu menggunakan model mesin Mealy telah dibangunkan. Model formal yang diekstrak digunakan untuk mengesan interaksi yang tidak berfungsi (kesilapan integrasi) atau masalah komposisi antara komponen bersepadu dalam sistem. Berdasarkan keputusan kajian, kaedah yang dicadangkan mempunyai kesan yang ketara dalam mengurangkan bilangan pertanyaan keluaran yang diperlukan untuk mengekstrak model rasmi komponen perisian bersepadu dan prestasinya adalah 50% lebih baik berbanding dengan kaedah yang sedia ada. Kedua, berdasarkan model formal yang telah diekstrak, teknik ujian integrasi berasaskan model (MITT) untuk CBSS telah dibangunkan. Akhirnya, keberkesanan MITT ditunjukkan dengan menggunakankannya dalam kajian tempahan makanan dalam penerbangan dan kajian kes lif, menggunakan tiga parameter penilaian. Keputusan kajian menunjukkan bahawa MITT berkesan dan mengatasi teknik Shahbaz dalam kajian tempahan makanan dalam penerbangan dan kajian kes lif. Dari segi komponen yang dikaji untuk kajian kes tempahan makanan dalam penerbangan dan kajian kes lif, masing-masing, keupayaan pengesanan ralat adalah 70.62% dan 75% bagi setiap kajian kes.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DECLARATION</td>
<td>ii</td>
</tr>
<tr>
<td></td>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGEMENT</td>
<td>iv</td>
</tr>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td></td>
<td>ABSTRAK</td>
<td>vi</td>
</tr>
<tr>
<td></td>
<td>TABLE OF CONTENTS</td>
<td>vii</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>xii</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>xiv</td>
</tr>
<tr>
<td></td>
<td>LIST OF ABBREVIATIONS</td>
<td>xvi</td>
</tr>
<tr>
<td></td>
<td>LIST OF APPENDICES</td>
<td>xviii</td>
</tr>
</tbody>
</table>

## 1 INTRODUCTION

1.1 Overview

1.2 Research Motivation

1.3 Statements of the Problem

1.4 Research Questions

1.5 Research Goal

1.6 Research Objectives

1.7 Significance of the Study

1.8 Scope of the Study

1.9 Thesis Outline

## 2 LITERATURE REVIEW

2.1 Introduction

2.2 CBSS Testing Levels

2.2.1 Unit Testing

2.2.2 Integration Testing

2.2.3 System Testing

2.2.4 Comparison

2.2.5 Comparison
2.3 Integration Testing of CBSS

2.3.1 Significance of Integration Testing in CBSD Life Cycle

2.3.2 Existing Approaches that Support Integration Testing in CBSS and the Classification

2.3.2.1 Built-in Testing Approach

2.3.2.2 Metadata Based Testing Approach

2.3.2.3 Testable Architecture Approach

2.3.2.4 Certification Strategy

2.3.3 Comparative Evaluation

2.3.3.1 The Evaluation Criteria

2.3.3.2 Comparative Evaluation Remarks

2.3.4 Drawbacks and Limitations of the Existing Techniques

2.4 Related Works on Model-Based Integration Testing

2.4.1 MBIT Based on UML

2.4.2 MBIT Based on UTP

2.5 The Approach of Learning and Testing

2.5.1 Active Automata Learning

2.5.1.1 Overview of the Learning Method $L^*$

2.5.1.2 Adaptations of the Angluin’s Method $L^*$

2.5.1.3 The Mealy Machine Methods $L_{M^*}$ and $L_{M^+}$

2.5.2 Applications and State-of-the-art Model Inference Integration Testing

2.6 Overview of the Most Important Active Automata Learning Tools

2.6.1 RALT

2.6.2 LearnLib

2.6.3 Libalf

2.6.4 Comparison

2.7 Learning Finite State Machine

2.8 Summary
3 RESEARCH METHODOLOGY

3.1 Introduction

3.2 Research Process

3.2.1 Research Process (1): Design and Implementation of the Proposed Model Extraction Method

3.2.1.1 Step 1: Development

3.2.1.2 Step 2: Evaluation

3.2.2 Research Process (2): Design and Implementation of the Proposed Integration Testing Technique

3.2.2.1 Step 1: Development

3.2.2.2 Step 2: Evaluation

3.2.3 Research Process (3): Evaluation and Comparison of the Proposed Integration Testing Technique

3.2.3.1 Step 1: Checking the Effectiveness of the Proposed Technique Using Case Studies

3.2.3.2 Step 2: Conducting A Comparative Analysis

3.3 Research Framework

3.4 Case Studies

3.4.1 Case Study 1: The HVAC Controller Case Study

3.4.2 Case Study 2: The Edinburgh Concurrency Workbench Case Study

3.4.3 Case Study 3: The Air Gourmet Case Study

3.4.4 Case Study 4: The Elevator Case Study

3.4.5 Comparison of the Case Studies

3.5 Summary

4 A METHOD TO EXTRACTING THE FORMAL FINITE STATE BEHAVIORAL MODEL

4.1 Introduction

4.2 Extracting the Mealy Machine Models

4.2.1 Preliminaries
4.2.2 The Proposed Mealy Models Extraction Method $L_M^x$

4.2.2.1 Observation Table in the Proposed $L_M^x$ Method 67

4.2.2.2 Handling Counterexamples in the Proposed $L_M^x$ Method 68

4.2.2.3 An Illustrative Example for Learning with $L_M^x$ 69

4.2.2.4 Comparison Between $L_M^x$ and $L_M^+$ 71

4.3 Evaluation of the Proposed Method $L_M^x$

4.3.1 Evaluating the Applicability of the Proposed Method $L_M^x$

4.3.1.1 Result of Extracting the HVAC Controller Using $L_M^x$ 74

4.3.2 Evaluating the Performance of $L_M^x$ Method Relative to Other Methods 76

4.3.2.1 Performance Metric 78

4.3.2.2 The Experimental Setting 78

4.3.2.3 The Experimental Results and Discussion 80

4.4 Summary 85

5 THE PROPOSED MODEL-BASED INTEGRATION TESTING TECHNIQUE 86

5.1 Introduction 86

5.2 Preliminaries 87

5.2.1 System Structure 87

5.2.2 Basic Definitions 88

5.3 The Proposed MITT in Details 89

5.3.1 The MITT Architecture 89

5.3.2 Detailed Description 92

5.3.2.1 C1: Inferring Approximated Model of Each Component 92

5.3.2.2 C2: Construct and Analyze Product 94

5.3.2.3 C3: Confirm or Relearning Models 96

5.3.2.4 C4: Test Generation 97


# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparing the main level of testing CBSS</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Summary of the strengths and weaknesses of the existing approaches</td>
<td>28</td>
</tr>
<tr>
<td>2.3</td>
<td>The criteria for evaluating existing techniques</td>
<td>29</td>
</tr>
<tr>
<td>2.4</td>
<td>Application of evaluation criteria to the existing integration testing techniques for CBSS</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Overview over the most important active learning tools</td>
<td>42</td>
</tr>
<tr>
<td>3.1</td>
<td>Research operational framework</td>
<td>55</td>
</tr>
<tr>
<td>3.2</td>
<td>The comparison of the case studies</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>Initial observation table</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Model inference of Mealy machine in Figure 4.6</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of $L_M^-$ and $L_M^+$</td>
<td>71</td>
</tr>
<tr>
<td>4.4</td>
<td>$L_M^-$ compared to $L_M^+$ for learning HVAC controller component</td>
<td>76</td>
</tr>
<tr>
<td>4.5</td>
<td>Experimental data of CWB examples</td>
<td>77</td>
</tr>
<tr>
<td>4.6</td>
<td>Results of learning models of CWB examples by the proposed method $L_M^-$ compared to $L_M^+$ method</td>
<td>81</td>
</tr>
<tr>
<td>4.7</td>
<td>Average and standard deviation for the number of output queries of the proposed method $L_M^-$ with $L_M^+$ and $L_M^*$ methods</td>
<td>84</td>
</tr>
<tr>
<td>6.1</td>
<td>The experimental data of the air gourmet case study</td>
<td>113</td>
</tr>
<tr>
<td>6.2</td>
<td>Results of learning the air gourmet case study</td>
<td>115</td>
</tr>
<tr>
<td>6.3</td>
<td>Results of output queries for the air gourmet case study</td>
<td>116</td>
</tr>
<tr>
<td>6.4</td>
<td>Average and standard deviation for the number of output queries of the air gourmet case study</td>
<td>118</td>
</tr>
<tr>
<td>6.5</td>
<td>Results of errors detected in the air gourmet case study</td>
<td>118</td>
</tr>
<tr>
<td>6.6</td>
<td>The experimental data of the elevator case study</td>
<td>121</td>
</tr>
<tr>
<td>6.7</td>
<td>Results of learning the elevator case study</td>
<td>122</td>
</tr>
<tr>
<td>6.8</td>
<td>Results of output queries for the elevator case study</td>
<td>124</td>
</tr>
</tbody>
</table>
6.9 Average and standard deviation for the number of output queries of the elevator case study 126
6.10 Results of errors detected in the elevator case study 126
A.1 Initial observation table for Mealy inference of the HVAC controller 153
A.2 The HVAC controller inference observation table after adding the suffix $T5$ to $S_M$ 154
A.3 The HVAC controller inference observation table after adding the suffix $T25$ to $S_M$ 155
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Recognized problems</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Levels of testing in CBSD life cycle (Rehman et al., 2007)</td>
<td>16</td>
</tr>
<tr>
<td>2.2</td>
<td>Classification of integration testing techniques of CBSS</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Learning and testing approach (Shahbaz, 2008)</td>
<td>36</td>
</tr>
<tr>
<td>2.4</td>
<td>Components of the $L^*$ method (Czerny, 2014)</td>
<td>38</td>
</tr>
<tr>
<td>3.1</td>
<td>Research processes</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>The methodology for developing and evaluating the proposed model extraction method $L_M^\times$</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>The methodology for developing and evaluating the proposed integration testing technique MITT</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td>Research framework</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>An example of Mealy machine</td>
<td>63</td>
</tr>
<tr>
<td>4.2</td>
<td>An example of DFA</td>
<td>63</td>
</tr>
<tr>
<td>4.3</td>
<td>Active learning approach</td>
<td>64</td>
</tr>
<tr>
<td>4.4</td>
<td>Flowchart of the proposed Mealy models extraction method $L_M^\times$</td>
<td>65</td>
</tr>
<tr>
<td>4.5</td>
<td>Pseudo-code for the proposed Mealy models extraction method $L_M^\times$</td>
<td>66</td>
</tr>
<tr>
<td>4.6</td>
<td>The method $L_M^+$ (Shahbaz and Groz, 2014)</td>
<td>66</td>
</tr>
<tr>
<td>4.7</td>
<td>Pseudo-code for treating the $CE$</td>
<td>68</td>
</tr>
<tr>
<td>4.8</td>
<td>Treating the $CE$ using Suffix1by1</td>
<td>68</td>
</tr>
<tr>
<td>4.9</td>
<td>Mealy machine</td>
<td>69</td>
</tr>
<tr>
<td>4.10</td>
<td>Global view of the HVAC system</td>
<td>73</td>
</tr>
<tr>
<td>4.11</td>
<td>Learning platform for RALT tool</td>
<td>75</td>
</tr>
<tr>
<td>4.12</td>
<td>Mealy machine model of HVAC controller component</td>
<td>75</td>
</tr>
<tr>
<td>4.13</td>
<td>Settings for learning CWB examples with RALT</td>
<td>78</td>
</tr>
<tr>
<td>4.14</td>
<td>Pseudo-code of the test driver</td>
<td>79</td>
</tr>
<tr>
<td>4.15</td>
<td>Pseudo-code of the oracle</td>
<td>80</td>
</tr>
<tr>
<td>4.16</td>
<td>Comparison results of the number of output queries of the proposed method $L_M^\times$ against $L_M^+$ method</td>
<td>82</td>
</tr>
</tbody>
</table>
4.17 Comparison results of the number of output queries of the proposed method $L_M^\times$ with $L_M^+$ and $L_M^*$ methods

4.18 Reduction in number of output queries of the proposed method $L_M^\times$ against $L_M^+$ and $L_M^*$ methods

5.1 The structure of system with $n$ components

5.2 Architecture of the proposed MITT

5.3 The procedure for extracting the formal finite state behavioral models of integrated software components

5.4 Constructing and analyzing product

5.5 Pseudo-code of the algorithm for generation of test cases in H-Switch Cover criterion (De Souza et al., 2015)

5.6 Original Mealy machine model

5.7 Creation of the dual graph from the original Mealy machine model

5.8 Balanced graph

5.9 An illustrative example of Mealy system CBS

5.10 The extracted Mealy machine model $M_A^{(1)}$ of component $A$

5.11 The extracted Mealy machine model $M_B^{(1)}$ of component $B$

5.12 The product of Mealy machine $\prod^{(1)}$ of component $A$ and $B$

5.13 The relearned Mealy machine model $M_B^{(2)}$ of component $B$

5.14 The refined product of Mealy machine $\prod^{(2)}$ of component $A$ and $B$

5.15 The relearned Mealy machine model $M_A^{(2)}$ of component $A$

5.16 The refined product of Mealy machine $\prod^{(3)}$ of component $A$ and $B$

6.1 The air gourmet case study

6.2 Percentage comparison of learning the air gourmet case study

6.3 Comparison results of the output queries of the air gourmet case study

6.4 Percentage comparison of errors detected in the air gourmet case study

6.5 The elevator case study

6.6 Percentage comparison of learning the elevator case study

6.7 Comparison results of the output queries of the elevator case study

6.8 Percentage comparison of errors detected in the elevator case study
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>Built-in Testing</td>
</tr>
<tr>
<td>CBSD</td>
<td>Component-Based Software Development</td>
</tr>
<tr>
<td>CBSE</td>
<td>Component-Based Software Engineering</td>
</tr>
<tr>
<td>CE</td>
<td>Counterexamples</td>
</tr>
<tr>
<td>CBS</td>
<td>Component-Based Software</td>
</tr>
<tr>
<td>CBSS</td>
<td>Component-Based Software Systems</td>
</tr>
<tr>
<td>CD</td>
<td>Component Developer</td>
</tr>
<tr>
<td>CDT</td>
<td>Component Deployment Testing</td>
</tr>
<tr>
<td>COTS</td>
<td>Components-Off-The-Shelf</td>
</tr>
<tr>
<td>CUL</td>
<td>Component Under Learning</td>
</tr>
<tr>
<td>CU</td>
<td>Component User</td>
</tr>
<tr>
<td>C+ BIT</td>
<td>Component+ Built-in Testing</td>
</tr>
<tr>
<td>CIG</td>
<td>Component Interaction Graph</td>
</tr>
<tr>
<td>CWB</td>
<td>Edinburgh Concurrency Workbench</td>
</tr>
<tr>
<td>DPE</td>
<td>Date Parsing Exception</td>
</tr>
<tr>
<td>DFA</td>
<td>Deterministic Finite-state Automata</td>
</tr>
<tr>
<td>EFSM</td>
<td>Extended Finite State Machine</td>
</tr>
<tr>
<td>ERTS</td>
<td>Embedded Real-Time Systems</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>GOSD</td>
<td>Graph of Sequence Diagram</td>
</tr>
<tr>
<td>HVAC</td>
<td>Heating, Ventilation, &amp; Air Conditioning</td>
</tr>
<tr>
<td>IAE</td>
<td>Illegal Input Exception</td>
</tr>
<tr>
<td>IIE</td>
<td>Invalid Input Exception</td>
</tr>
<tr>
<td>IT</td>
<td>Independent Tester</td>
</tr>
</tbody>
</table>
LTS – Labeled Transition Systems
MBT – Model-Based Testing
MBIT – Model-Based Integration Testing
NFA – Non-deterministic Finite Automata
NFE – Number Format Exception
NPE – Null Pointer Exception
OCL – Object Constraint Language
ONFSM – Observable Non-deterministic Finite State Machines
PFSM – Parameterized Finite State Machines
RF – Reduction Factor
RALT – Rich Automata Learning and Testing
SCT – Software Component Testing
STECC – Self-TEsting COTS Components
SUL – System Under Learning
SUT – System Under Test
TFM – Transaction Flow Model
UML – Unified Modeling Language
UTP – UML Testing Profile
UTL – Unable To Launch
## LIST OF APPENDICES

<table>
<thead>
<tr>
<th>APPENDIX</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Learning the HVAC controller component</td>
<td>153</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Overview

Since the early 1990’s, software development community was encountering various challenges and issues in developing software applications (Sneed, 2010). Today’s modern software applications have become increasingly larger in scale and inherited several complex characteristics (Chakraborty and Chaki, 2016; Ghazi et al., 2015; Lity et al., 2015). Current software applications are composed of several sub-systems, more complicated, evolve over time into different versions, exist in many different variants, distributed amongst network and very critical (Di Ruscio et al., 2014; Lochau et al., 2014; Tran et al., 2015). Consequently, this complex characteristic makes the development of modern software systems very expensive (Guan and Offutt, 2015). At the same time, software production cost and time-to-market for developing and delivering software applications need to be reduced due to the customer demands and the current competition amongst software businesses or companies to meet market demands (Kajtazovic et al., 2014; Muschevici et al., 2015). Hence, achieving the goals of on-time delivery and quality becomes more challenging (Alégroth et al., 2015; Kaur and Batolar, 2015; Meyerer and Hummel, 2014). On the other hand, as today’s software applications become more and more complex over time in terms of size, effort and cost, software quality accordingly have become increasingly important (Elhag et al., 2013; Farjaminejad et al., 2014; Goeb and Lochmann, 2011). Given these reasons, to cope with the challenges that are being faced by software community, software engineers and developers try to look for innovative alternative approaches that facilitate the development of current complex and very critical software applications (Bui, 2005; Mahmood et al., 2015; Patel et al., 2012).

Consequently, software engineers and developers found that today’s large, complex, high quality software applications could be efficiently developed partially
if not completely, by reusing and integrating pre-built, pre-tested, high-quality, well-defined independent and “plug and play” sub-systems denoted by “software components” that their operations have been previously tested as a part of successful applications (Kaliraj et al., 2014; Siddiqui and Tyagi, 2016). This idea gave birth to a very cost-effective, attractive, fast and efficient research branch in the area of software engineering known as Component-Based Software Engineering (CBSE) (Le and Pham, 2012; Tarawneh et al., 2011). Therefore, building a software applications from prefabricated small pieces of software parts or components is seen as a solution to these problems (Dimri, 2015; Kaur and Tomar, 2015). As a consequence, Component-Based Software Systems (often referred as CBSS or CBS) have become a heart of most modern software applications. It has been a common trend in system development nowadays and the key benefits introduced by the CBSE approach (Pramsohler et al., 2015; Shu-Fen et al., 2010). Hence, CBSE paradigm has introduced significant changes in the development of current modern complex software applications thanks to their solid advantages (Tiwari and Chakraborty, 2015; Tran et al., 2015).

CBSE approach has a great impact in the last few years in a wide variety of application areas such as business applications, automotive and telecommunications, distributed control applications, web-based applications, scientific applications, medical and healthcare applications, and others, including several types of Embedded Real-Time Systems (ERTS) (Guan and Offutt, 2015; Orso and Rothermel, 2014; Tao et al., 2015; Zaki et al., 2015). These current software systems require to ensure high degree of quality, reliability and security as well as safety. Therefore, any bug and error in these applications can affect and do the serious damage to the economies, businesses, environment as well as loss of lives. Hence, testing becomes one of the important activities and a fundamental task of software engineering is linked with the development effort of any software application with the purpose of finding faults (Ahmed and Ibrahim, 2015; Elghondakly et al., 2016; Lachmann et al., 2015). Unfortunately, testing today’s software applications in general is an expensive activity in software development life cycle in terms of time and budget as well as other resources (Bertolino, 2007; Mahmood, 2011).

Literature suggests that 30% to 60% of the development time of a software product is dedicated to testing (Afzal et al., 2016; Brar and Kaur, 2015; Ellims et al., 2006; Harman et al., 2015; Mohi-Aldeen et al., 2017). In spite of that, it is a widely used methodology and essential stage used to evaluate the functionality of software applications, increases the confidence of the developers in the reliability and correctness of software when it will be released (Moiz, 2017). It also improves quality
of operation during deployment environment, by revealing errors and failures in order
to produce high-quality software applications, especially for systems being developed
by integrating prefabricated “plug and play” subsystems or software components.

Nowadays, integration of software components is a common trend and major
technique of modern software development (Castro and Francisco, 2013; Groz et al.,
2015). Therefore, in CBSE several components, often pre-built and third-party
components known as Components-Off-The-Shelf (COTS) coming from outside have
to be integrated together in order to build a CBSS, instead of developing the systems
from scratch (Andreou and Papatheocharous, 2016; Groz et al., 2008; Shahbaz and
Groz, 2014; Verma, 2012). In spite of the reality that a software component might
go through different levels of testing, unit testing still cannot ensure the behavior and
reliability of software component after integration in a new environment (Brohi and
Jabeen, 2012; Gupta, 2015; Mahmood et al., 2007). Therefore, the system developers
should check to ensure that the components developed separately should work properly
when integrated. Furthermore, to guarantee the correct functionality of the system,
a wide number of possible interactions between integrated software components in
the system may need to be tested (Guan and Offutt, 2015). Additionally, many
faults may not be obvious until integration and some complex behaviors can only be
observed when related components are integrated (Holling et al., 2016). Thus, testing
each component independently does not eliminate the need for integration testing.
Therefore, integration testing, which bridges component unit testing and component
system testing, plays an important role in the testing process of Component-Based
Software Development (CBSD) life cycle (Sirohi and Parashar, 2013).

In integration testing phase, the individual software components are assembled
and verified as a group to attain a high level of quality and reliability (Belli et al., 2009;
Khan and Nadeem, 2013). Moreover, integration testing is essential level of quality
assurance that minimizes the risk of the system not working effectively and efficiently,
and focuses on the prevention of integration bugs (Khan and Singh, 2012). Therefore,
it plays a substantial part in detecting faults during a CBSD life cycle (Mahmood,
2011). Approximately, 40% of the software errors are discovered and revealed during
integration testing (Kaur et al., 2011). On the other hand, integration testing of CBSS is
most time consuming and expensive part of the testing process in CBSD, and received
significantly little attention from practitioners and researchers in the field of software
engineering (Bai et al., 2001; Gao et al., 2003; Ning et al., 2013; Shashank et al., 2010;
Shukla and Marwala, 2012).
1.2 Research Motivation

When considering integration testing of CBSS, many issues and challenges could be identified specially when considering integrated black-box software components (Bertolino and Polini, 2003; Beydeda and Gruhn, 2003b; Khan et al., 2011; Machado et al., 2007; Reza and Cheng, 2012). In most cases, the source code, detailed documentations, design specifications, and formal models of integrated components are missing or not available to the system designers, making system integration and integration testing very time consuming and more quite challenging (Haser, 2015). The consequences of these challenges, several works have been proposed during the last decade, aiming to facilitate integration testing of CBSS. Here in the following sections the issues and challenges that are related to the scope of this thesis directly or indirectly are summarized, and the efforts made by researchers to tackle it.

The first challenge deals with a lack of information exchanged between component producers and component consumers. Component developers and component users need to exchange different types of information during the development of the component itself and during the development of CBSS in order to facilitates the testing process, and to develop high-quality and reliable CBSS. Unfortunately, unavailable information limits the capacity of both component developers and component users or even the independent tester (third-party tester) to test candidate components efficiently. Therefore, there is an effort made by several researchers to aggregate valuable information to the component in order to facilitate the integration testing activities at the component developers’ side and to minimize as much as possible the dependence of component users and the independent tester (third-party tester) on the information provided by the component providers.

In 1998, Liu and Richardson (1998) proposed a technique to capture information related to the usage of the component inside source code snippets. In 1999, Harrold et al. (1999) initiated the idea of metadata by proposing a technique for analyzing and testing CBSS from two perspectives of component providers and component users. In 2001, Orso et al. (2001) suggested another category of summary information to support component testing called component metadata. Component users can then use the information provided by the metadata to test and analyze a CBSS. Edwards (2001) proposed the reflective wrapper technique that analyzes the best way to package specification and verification information of formally specified components for distribution to component consumers. In 2003, Bertolino and Polini
(2003) proposed component deployment technique for component users to facilitate the testing within a user’s target environment of a component independently developed. Silva et al. (2009) presents a technique covered by a CASE tool integrated in the development environment to support components integration testing aiming to reduce the lack of information between component producers and component consumers. In 2010, Naseer et al. (2010) presented a technique to use metadata technique for CBS black box testing and developed a tool which takes <.dll> component. Brohi and Jabeen (2012) proposed a technique that enhances component testability and to facilities the integration testing by defining a uniform information flow in the component life cycle.

Indeed, all the proposed previous related research that follows in this category, however, don’t handle heterogeneous components and have an effect on the implementation transparency of the component, which is an important characteristic in software component.

Unavailability of the components source code, or internal workings (structure) of the integrated components might not be accessed by a component users and the independent tester (third-party tester) due to organizational or legal restriction, will affect and reduce the controllability and testability of candidate components, and hence the overall integration testing of CBSS. Consequently, a number of proposed solutions provided by several researchers to equip a software components with a specific testable architecture that permits component users and the independent tester (third-party tester) to execute test cases easily in order to support self-testing and to facilitate the integration testing process.

In 1999, Wang et al. (1999) proposed the idea of Built-in Testing approach (BIT) to increase component testability. BIT based on the development of test cases in component source code as additional member functions with the normal member functions. Consequently, Atkinson and Grob (2002) presented an example of BIT technique, called component+. It is related to contract testing in model-driven component-based development. In the same year, Gao et al. (2002) proposed the testable beans technique. The idea of BIT was extended by Mahmood (2011) to present a component-based software integration testing technique to prioritize test cases and identify integration test criteria using software complexity measures.

In 2001, Martins et al. (2001) has presented a technique for Self-testable software component. The proposed technique is another example to add extra
information to software component intentionally to improve component testability by integrating testing resources into it. Concat tool was developed to support the proposed technique. In 2003, Beydeda and Gruhn (2003a) presented the Self-Testing COTS Components (STECC) technique that augment components with functionality specific to testing tools that is capable of conducting some or all activities of the component user’s testing processes.

Accordingly, due to the unavailability of integrated software component source code, and with the lack of component information, detailed documentations, complete functional behavior descriptions and design specifications for analysis and testing the systems of integrated components, specifically when one considers black-box software components, existing integration testing techniques under this category has become ineffective to handle and detect erroneous interactions of components during the integration testing phase, and fail to test today’s modern complex CBSS. Additionally, in most cases the formal models for analysis and testing the integrated components, and which will help to understand their possible behaviors in the system are always missing or not available to the system designers. Furthermore, there could be a large number of possible interactions between integrated software components which are undesirable and which could affect the function of each others, hence, these interactions may need to be tested to ensure the correct functionality of the CBSS. As a result, some complex behaviors are not observed until related components are integrated and many faults may not be visible until integration, making integration testing very quite challenging by the existing integration testing techniques and most time consuming task. Moreover, this invisible behavior of a component can affect the behavior of the overall CBSS.

The Unified Modeling Language (UML) and its diagrams are commonly used and gained wide acceptance amongst researchers to address the necessity for additional information by appending some UML models with software components. Accordingly, the techniques consist of UML models such as (Barisas et al., 2013; Gallagher et al., 2006; Hartmann et al., 2000; Kaur et al., 2011; Machado et al., 2007; Mussa and Khendek, 2012; Shang and Zhang, 2006; Wu et al., 2003; Zheng and Bundell, 2008, 2007a,b) have been proposed in order to facilitate the integration testing process at the component user side.

However, the drawbacks of UML based integration testing approach is that, with the help of reverse engineering its possible to modify a component and to access the component source code by using some UML reverse engineering tools. Furthermore, these UML models is unrealistic because COTS evolve along
time to incorporate additional requirements, which quickly invalidates the original project (Castro and Francisco, 2013). Moreover, the techniques consist of semi-formal models such as UML models affecting the implementation transparency of software components. Additionally, these semi-formal models is not sufficient for the component users to understand its behavior completely, and it cannot be used as input to the existing MBT approaches. Hence, the traditional techniques for MBT approaches such as static analysis, program slicing, invariant detection, model extraction, validation and verification also has become ineffective when one considers systems of integrated black-box software components.

To conclude, despite several existing proposals, the existing works discussed in this section of adding additional structure for reliable use of component applications, and/or adding information with software components to facilitate the integration testing process suffers from a number of drawbacks and limitations, leading to ineffective testing and, ultimately, to poor software quality. These drawbacks and limitations are very important and should be resolved. Hence, new solutions have to be developed to cope with these drawbacks and limitations. The drawbacks and limitations of the existing works that are directly related to the scope of this thesis are highlighted here:

i. Difficulty of understanding and describing the behaviors of integrated components, due to the frequent lack of information and/or implementation details. Moreover, the formal models of integrated components that can be used for analysis, testing and documentation, and which will help to understand their possible behaviors in the system are always missing or not available to the system designers.

ii. It is impossible to use it in some cases, for instance, when there is no formal model to understand the possible behavior of the integrated components. A key problem, however, is the construction of models that describe the intended behavior of the integrated components in a system.

iii. Difficulty of analyzing and validating the integrated components, due to the restricted access of components source code, detailed documentations is not sufficient to solve details about its interaction with other components, absence of components models to check the possible interaction between the components, and other development information to the system designers.
iv. Lack of accurate and effective formalism for information representation (internal structure of a component is generally unknown). Hence, how to query and retrieve test information effectively has become the key problem while reusing test information. Due to that, the interpretation of information is not clear due to it is non-uniformity, hence requires understanding the representation prior to interpreting the meaning.

v. Some techniques expose component implementation by providing semi-formal models such as metadata, BIT, and UML models, allowing reverse engineering to access the source code of software component, thus affecting the implementation transparency of software components. Furthermore, the implementation transparency property raises some difficulties when the CBSS is to be tested. Moreover, these informal models cannot be used as input to the existing Model-Based Testing (MBT) approaches, which is a new technique which rely on explicit or accurate models for testing purposes, and aims to make testing more effective and more efficient.

In view of the above background, and in order to address the important challenges and limitations of the existing proposals, this thesis aimed at proposing an integration testing technique for CBSS, by exploits the use of learning and testing approach for integration testing of CBSS. Therefore, the proposed technique in this thesis combining model learning and testing techniques for testing of a system of integrated software components.

1.3 Statements of the Problem

Engineering high-quality CBSS is essential for the involved enterprises and demands interest from both academia and industry. With the increasing complexity of today’s modern CBSS, verification and validation techniques are becoming more and more important. Therefore, integration testing has become a very essential activity linked with the development effort of any software applications and most important means to ensure the quality of today’s modern CBSS. Unfortunately, nowadays many difficulties arise in integration testing of CBSS leads to new challenges, and it has become a more and more complex task that significantly influenced by a number of factors. Even so, integration testing of CBSS has been one of the open research area that is rarely investigated and received significantly little attention from practitioners and researchers in the field of software engineering, and remains comparatively
less well-solved. Therefore, it requires the support of tools, methodologies and well established techniques to mitigate the challenges and limitations of the existing works, and to support the development of high-quality CBSS. On the other hand, understanding the behavior and testing the integrated software components is another challenging task, due to the unavailability of their source code, updated specifications or formal models. Moreover, obtaining the accurate formal models for existing software components, which precisely describe the behavior of the integrated software components is still an open and interesting problem.

The recognized problems of the integration testing of CBSS are presented in Figure 1.1. However, by understanding the problem background which has been discussed in the previous section, it can be concluded that continues efforts and further works still required, and a new solution for improvement should be developed to mitigate the above limitations and gaps left by past research works effectively in order to enhance the development of today’s modern CBSS.

![Integration testing of CBSS](image)

**Figure 1.1: Recognized problems**
The environment of a CBSS can be seen as a form of black-box (service interface). Therefore, a model-based approaches such as MBT can be seen as potential solution. Since MBT is a black-box approach, it tend itself to be suitable mechanism to deal with CBSS problems. Accordingly, among the existing testing techniques, MBT is a promising candidate to be applied in the context of integration testing of CBSS. Despite the amount of literature on integration testing, model-based integration testing techniques are quite limited (Dias Neto et al., 2007). MBT has become a common trends which have added many values to the engineering of software projects. MBT has more advantages, and can well support component integration testing. Besides the automatic test case generation, another relevant characteristic for testing components is the adoption of formal models and black-box testing strategy (Haser, 2015). Black-box testing are appropriate for integration testing of CBSS because internal structures of integrated software components are always missing or not available to the system designers, the complexity of interactions and test harness can be abstracted, the formality of the model contribute to more reliable tests. Furthermore, several benefits such as (high level of automation, reducing cost and time for testing, high fault detection rate and generating tests automatically) were obtained from the adequate application of MBT to software systems.

### 1.4 Research Questions

This research intends to propose an integration testing technique for CBSS in order to mitigate above mentioned problems. Despite the powerful features of MBT approach, most existing techniques provide only limited support for integration testing. This leads to the main research question to be answered in this study:

"**How is it possible to use MBT to develop an effective model-based integration testing technique for CBSS?**"

To address the primary research question given above, it is further broken down into sub-questions.

**RQ1:** How to understand the behavior of integrated software components in the system?

**RQ2:** How to develop an effective integration testing technique for CBSS in order to overcome the identified challenges?
RQ3: How to measure the effectiveness of the proposed technique?

1.5 Research Goal

This research work concentrates on the problem of testing the integrated software components in the system in the missing of their formal behavioral models. Given a set of black-box software components that are integrated in a system, the first major goal of this study is to infer/extract the formal models which describes the behaviors of integrated software components, by proposing a method to infer the approximated finite state behavioral models in term of finite state machine (Mealy machines), that represents the precise description of the intended behaviors of the integrated components formally directly from the components, using the idea of active learning approach. Then, the second major goal of the study is to propose an effective model-based integration testing technique, which combines model learning and testing techniques, to identify the faulty interaction between the integrated components in a system based on the extracted formal models and with the help of learning and testing approach.

1.6 Research Objectives

To achieve the goal, the following four objectives need to be undertaken with the aim of finding answers to the research questions:

i. To propose a method that extracts the formal finite state behavioral models of integrated software components using active learning approach and benchmark the performance based on the number of output queries.

ii. To develop an effective integration testing technique for CBSS using the extracted models and with the help of learning and testing approach.

iii. To demonstrate and measure the effectiveness of the proposed technique by applying the proposed technique in the selected applications as case studies, using three evaluation parameters, namely learned components, output queries, and error detection capability.

iv. To evaluate and compare the proposed technique against Shahbaz technique based on learned components, output queries, and error detection capability.
1.7 Significance of the Study

The significance of this study is to mitigate the limitations and gaps left by past research works in integration testing of CBSS, by proposing an effective integration testing technique of CBSS, which combines model learning and testing techniques for testing of a system of integrated black-box software components. Thus, this thesis exploits the use of learning and testing approach for integration testing of CBSS. The study proposes solutions into two directions:

i. **Reverse engineering:** Understanding the behaviors of the integrated black-box software components, by deriving (extracting) the formal models of the components. In this study, the software components are learned in order to extract their formal finite state behavioral models as Mealy machine models.

ii. **Validation:** Developing an effective model-based technique for integration testing of CBSS. Thus, the integrated software components is tested and analyzed using their learned formal models (Mealy machine models). In this study, the use of components formal models will help in revealing compositional problems or faulty interactions (integration bugs) between integrated software components and general errors in the system.

1.8 Scope of the Study

The scope of this research work has been limited to the following aspects:

i. **Integration testing:** As discussed before in this chapter, in the CBSD life cycle, three basic kinds of testing are needed in order to detect and reveal errors, namely “unit testing (component testing), integration testing (deployment testing), and finally system testing”. A brief description of this three levels of testing will be introduced in the next chapter. However, this research is concerned only on integration testing in the context of CBSS, and does not cover the other testing levels. Thus, the issues related to other testing levels are not dealt with in this study.

ii. **Components are black boxes:** The integrated components in the system may have different levels of exposure depending upon how much information about them is available. In literature, the terms black box, gray box and white box are used with reference to different levels of closure of the component internal
essence. This study considers that all components are black boxes, i.e., their functional specifications and implementation details are not available. However, the basic set of input symbols that can be given to a component through it is input interfaces are known, and for each input, the corresponding output of the component can be observed through it is output interfaces.

iii. **Modeling level**: The component exhibits regular behaviors, i.e., the component can be modeled as a finite state machine (Mealy machine). This study intends to learn only the behaviors prescribed by the control structure of the finite state model. Moreover, the study do not assume to know the upper bound on the number of states in the components. Instead of hunting for exact learning, the study aims to learn approximate models that are expressive enough to provide powerful guidance for testing and to enhance the behavior understanding of the integrated software components, and thus, of the system.

iv. **Focus on functional aspects**: This study focuses on behavior learning and studying the interactions between the integrated components and their functional aspects in the system. Therefore, the study are not dealing with other details, for instance, security, timing, and performance issues in the system.

v. **Case studies and their assumptions**: The proposed research work in this thesis has been validated using four different case studies that large enough to get some interesting results. Therefore, different case studies from different domains have been used in this research. Furthermore, in order to check the effectiveness of the proposed technique and to compare its results with the existing proposal’s results, the selected case studies are fully developed based on CBSD.

vi. **Benchmarking with existing techniques**: To the best of our knowledge, Shahbaz and Groz (2014) is the only work found in the literature that uses learning and testing approach for CBSS. Therefore, the technique proposed by Shahbaz and Groz (2014) is the best and closest work to compare its experimental results with the obtained experimental results of the proposed technique in this thesis.

### 1.9 Thesis Outline

This thesis is organized in seven chapters. The structure of these chapters as follows:
Chapter 2, Literature Review. This chapter is associated with literature review. First, the chapter provides information about testing CBSS. Then, a comprehensive review of integration testing of CBSS will be provided in details. Next, the chapter presents a discussion on the related work on model-based integration testing. Moreover, a detailed description of learning and testing approach will be provided. Next, an overview of the most important active learning tools will be given. Learning finite state machine will be highlighted also in this chapter.

Chapter 3, Research Methodology. This chapter explains comprehensively the research methodology used in this thesis in order to show how the objectives are achieved. This includes the research process and activities involved in each phase to depicts the flow of research step by step, and the research framework to show the main parts and components of the research. At last, the description and comparison of case studies that will be used in this research will be provided in this chapter.

Chapter 4, A Method to Extracting the Formal Finite State Behavioral Model. This chapter describes in details the proposed finite state behavioral model extraction method. In addition, this chapter also explains and discusses the results related to the experimental evaluation of the proposed method using case studies.

Chapter 5, The Proposed Model-based Integration Testing Technique. In this chapter, an integration testing technique for testing the integrated black-box software components in a system using approximated (partial) models of software components is proposed. Precisely, this chapter presents the structure and development steps of the proposed technique. The discussion includes all the necessary elements that related to the proposed technique. An illustrative example will be used to clarify the implementation of the proposed technique.

Chapter 6, Evaluation and Comparison of MITT. This chapter discusses the results of evaluating and comparing the proposed technique with other current techniques in order to explain the strengths and weaknesses related to the proposed technique. Therefore, the chapter explains the evaluation of the proposed technique in details. In particular, the experimental results are provided and discussed in details.

Chapter 7, Conclusion and Future Work. This chapter concludes the thesis by highlighting the summary, achievements of research objectives covered in this thesis, the contributions of research, and finally, the future work are elaborated.
REFERENCES


